Increasing the Robustness of Boosting Algorithms within the Linear-Programming Framework *

Yijun Sun[†,§], Sinisa Todorovic[‡], and Jian Li[§]
[†] Interdisciplinary Center for Biotechnology Research
[§] Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32610
[‡]Beckman Institute, University of Illinois at Urbana-Champaign,
405 N. Mathews Ave, Urbana, IL 61801, USA

Abstract

AdaBoost has been successfully used in many signal classification systems. However, it has been observed that on highly noisy data AdaBoost easily leads to overfitting, which seriously constrains its applicability. In this paper, we address this problem by proposing a new regularized boosting algorithm LP_{norm2}-AdaBoost (LPNA). This algorithm arises from a close connection between AdaBoost and linear programming. In the algorithm, skewness of the data distribution is controlled during the training to prevent outliers from spoiling decision boundaries. To this end, a smooth convex penalty function (l_2 norm) is introduced in the objective function of a minimax problem. A stabilized column generation technique is used to transform the optimization problem into a simple linear programming problem. The effectiveness of the proposed algorithm is demonstrated through experiments on many diverse datasets.

Keywords: pattern classification, large margin classifier, AdaBoost, linear programming, minimax problem, soft margin, regularization.

Journal of VLSI Signal Processing Systems
Submitted in April 2006

*Please address all correspondence to: Dr. Yijun Sun, Interdisciplinary Center for Biotechnology Research, University of Florida, P.O. Box 103622, Gainesville, FL 32610-3622, USA. E-mail: sun@dsp.ufl.edu.

1 Introduction

AdaBoost is a method for improving the accuracy of a learning algorithm (a.k.a. base learner). This is achieved by iteratively calling the base learner on re-sampled training data, and by combining the so-produced hypothesis functions together to form an ensemble classifier [1, 2]. AdaBoost has been successfully implemented in many signal processing systems [3, 4, 5, 6, 7, 8]. An extensive experimental evidence shows that AdaBoost rarely suffers from overfitting problems in low noisy regimes [2, 9, 10]. Recent studies on highly noisy patterns, however, have clearly demonstrated that overfitting in AdaBoost can occur [9, 10], which is a critical factor that constrains a wider applicability of AdaBoost. In this paper, our main goal is to address this problem.

In general, there are two distinct overfitting cases: (i) when a simple base learner (e.g., stump) or an unstable base learner (e.g., C4.5 [11]), and (ii) when a powerful base classifier (e.g., radial basis functions–RBF) is used. The analysis of the asymptotic behavior of AdaBoost in the first case shows that after a large number of iterations, the testing error may start to increase, despite the continuing increase in the margin of an ensemble classifier [12]. In the second case, AdaBoost quickly leads to overfitting only after a few iterations. These observations indicate that a regularization scheme is needed for AdaBoost in noisy settings.

One method to alleviate the overfitting of AdaBoost is to choose a simple base learner, and implement an early stop strategy. However, in this case, AdaBoost may not outperform a single well-designed classifier, such as RBF and support vector machine (SVM), which renders the use of AdaBoost unjustified. The second case of overfitting, when a strong base learner is used in AdaBoost, has not been to date well treated in the literature. Only a few algorithms have been proposed to address the problem, among which $\text{AdaBoost}_{\text{Reg}}$ achieves the state-of-the-art generalization results on noisy data [10]. The basic idea is to control the tradeoff between the margin and the sample influences to achieve a soft margin. Although, in comparison with other regularized algorithms, $\text{AdaBoost}_{\text{Reg}}$ empirically shows the best performance [10], it is not clear whether it converges, nor what its actual optimization problem is, since the regularization in $\text{AdaBoost}_{\text{Reg}}$ is introduced on the algorithm level [2, 13].

In this paper, we present a new regularized boosting algorithm aiming to improve the performance of a strong base learner. Our work is directly motivated by a close connection between AdaBoost and linear programming (LP). This connection was first noted in [14, 15], and was used in [12] to derive a new LP based boosting algorithm, referred to as LP-AdaBoost. LP-AdaBoost directly maximizes the classifier margin given a finite set of hypothesis functions. Empirical analyses of LP-AdaBoost show that the algorithm is capable of achieving a larger classifier margin than AdaBoost, as expected. However, LP-

AdaBoost almost always yields a worse performance than AdaBoost. This can be explained that LP-AdaBoost produces a classification scheme, which optimizes performance in the "worst case". That is, LP-AdaBoost maximizes the smallest sample margin regardless of the fact that data may not be separable. Thus, when the data distributions are highly overlapped (due for example to mislabeling present in the training set), LP-AdaBoost can be easily misled by a few outlier data samples, resulting in a suboptimal performance. One possible approach to alleviating this problem is to introduce a regularization scheme in LP-AdaBoost, which would control the skewness of the training-data distribution. Thereby, base learners would be disallowed to allocate all its resources onto several "hard-to-learn" samples, which typically turn out to be outliers.

The aforementioned idea is further explored in [10], where the original AdaBoost algorithm is first used to generate a predefined number of hypothesis functions, and thereby a linear programming problem is formulated. Slack variables are introduced into the optimization problem in the primal domain to achieve a soft margin, similar to SVM for non-separable data cases, discussed in [16]. By pursuing a soft margin instead of a hard margin, the resulting algorithm does not attempt to classify all training samples according to their class labels, but allows for a small training error. A more general algorithm is proposed in [17], where, instead of using AdaBoost for probing a hypothesis space with a possibly infinite number of members, a column generation technique is used to iteratively generate hypothesis functions when needed. Both algorithms can been interpreted in the dual domain as a strategy to constrain the data distribution into a box, and hence can be viewed as a penalty scheme on the distribution skewness using a hard limited type of penalty function. In this paper, we only consider the column-generation based algorithm, since it is more theoretically appealing than the one proposed in [10]. We refer to the algorithm as LP_{reg}-AdaBoost.

In this paper, we consider controlling the data distribution skewness by a smooth convex penalty function, specified as an additional term of the objective function in the original minimax formulation of LP-AdaBoost. This leads to a piecewise convex optimization problem. We use a stabilized column generation technique to iteratively generate the columns of a gain matrix and linearize the optimization problem into a simple LP problem. Unlike $\text{AdaBoost}_{\text{Reg}}$, the proposed algorithm that we call LP_{norm2}-AdaBoost has a clear underlying optimization scheme, and can be shown to converge in a finite number of iterations. Empirical results over a wide range of data demonstrate that our algorithm achieves a similar, and in some cases significantly better classification performance than $\text{AdaBoost}_{\text{Reg}}$.

The remainder of the paper is organized as follows. First, in Section 2 we present a brief review of AdaBoost. In Section 3 we study the connection between the minimax optimization problem and AdaBoost. Based on these discussions, a new algorithm, referred

to as LP_{norm2}-AdaBoost, is proposed. To demonstrate the effectiveness of the proposed algorithm, in Section 5, the results of a large-scale experiment on several artificial and real-world datasets are reported. We finally conclude the paper in Section 6 with our remarks.

2 AdaBoost

In this section, we give a brief review of AdaBoost. Suppose we have a training data set $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N \in \mathcal{R}^l \times \{\pm 1\}$, where $\mathbf{x}_n$ is a data-feature vector, and y_n is its associated label. Given a class of hypothesis functions $\mathcal{H} = \{h(\mathbf{x}) : \mathbf{x} \to \{\pm 1\}\}$, also called base learners, we are interested in finding an ensemble function $F(\mathbf{x})$ constructed as:

$$F(\mathbf{x}) = \sum_t \alpha_t h_t(\mathbf{x}) \qquad \text{or} \qquad f(\mathbf{x}) = \frac{1}{\sum_t \alpha_t} F(\mathbf{x}) \tag{1}$$

to improve the classification accuracy of the base learners. Both the combination coefficients $\boldsymbol{\alpha} = \{\alpha_t\}_t$ and the hypothesis functions $h_t(\mathbf{x})$ are determined in the learning process. Several ensemble methods [1, 18, 19] have been developed in the past few years, among which AdaBoost is the most popular. It is generally considered a first step towards practical boosting algorithm development. The pseudocode of AdaBoost is presented in Fig. 1. For more detailed description, the interested reader is referred to [2] and the references therein.

The main idea of AdaBoost is to repeatedly apply a base learning algorithm to the re-sampled versions of training data to produce a collection of hypothesis functions that are finally combined via a weighted linear vote to form the final decision. Each data pattern, $\mathbf{x}_n$, is associated with a weight, $d^{(t)}(n)$, in the t-th iteration. The weights are normalized over the dataset so that they form the data distribution, $\mathbf{d}^{(t)} = \{d^{(t)}(n)\}_n$, which changes in every training iteration. An intuitive idea in AdaBoost is that misclassified patterns in a given training iteration are associated with larger weights in the subsequent iteration (see Eq. (3)). Thereby, the base learner focuses more on those harder cases, for instance, the patterns near the decision boundary.

From Eq. (2), which is used to compute the hypothesis combination coefficients, it can be shown that AdaBoost exponentially reduces the training error to zero as the number of combined classifiers increases. However, driving the training error to zero does not guarantee that the final classifier can generalize well on unseen test patterns. One may even suspect that AdaBoost quickly leads to overfitting. However, there is a growing body of empirical evidences that show that AdaBoost effectively reduces the generalization error, and in many cases the generalization error continues to decrease even after the training error reaches zero.

The impressive generalization capability of AdaBoost has been extensively investigated both experimentally and theoretically [12, 20, 21, 22, 23]. One leading explanation is

AdaBoost

Initialization: $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, set the maximum iteration number T, set the data distributions $d^{(1)}(n) = 1/N$, for $n = 1 : N$

for $t = 1 : T$

1. Train base learner with respect to data distribution $\mathbf{d}^{(t)}$ and get hypothesis $h_t(\mathbf{x}) : \mathbf{x} \to \{\pm 1\}$.
2. Calculate weighted training error ϵ_t of h_t:

$$\epsilon_t = \sum_{n=1}^{N} d^{(t)}(n) I(y_n \neq h_t(\mathbf{x}_n))$$

where $I(\cdot)$ is the indicator function.

3. Compute combination coefficient:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{2}$$

4. Update weights:

$$d^{(t+1)}(n) = d^{(t)}(n) \exp\left(-\alpha_t y_n h_t(\mathbf{x}_n)\right) / C_t \tag{3}$$

where C_t is the normalization constant such that $\sum_{n=1}^{N} d^{(t+1)}(n) = 1$.

end

Output : $F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

Figure 1: The pseudocode of AdaBoost.

the margin theory [23], stating that AdaBoost asymptotically maximizes the margin of the resulting ensemble classifier, defined as $\rho = \min_{1\leq n\leq N} y_n f(\mathbf{x}_n) = \min_{1\leq n\leq N} \rho(\mathbf{x}_n)$. The problem of maximizing the classifier margin can be solved exactly by using linear programming:

$$\begin{array}{ll} \max_{(\rho,\boldsymbol{\alpha})} & \rho \ , \\ \text{s.t.} & \rho(\mathbf{x}_n) \geq \rho, \quad n = 1, \cdots, N \ , \\ & \|\boldsymbol{\alpha}\|_1 = 1, \boldsymbol{\alpha} \geq 0 \ . \end{array} \tag{4}$$

In the recent paper [24], however, the equivalence of the two algorithms has been proven to not always hold. Nevertheless, these two algorithms are closely connected in the sense that both algorithms try to maximize the classifier margin. Throughout this paper, we will make use of this connection to devise new AdaBoost-like algorithms.

3 Regularized LP Boosting Algorithms

In this section, we propose a novel regularized AdaBoost algorithm. To this end, we first discuss the relationship between AdaBoost and the minimax problem. The connection between the well-known minimax problem [25] and AdaBoost was first noted in [14, 15], and was used to determine the maximum margin that one can achieve given a hypothesis class by exploiting the dual relationship in linear programming. For simplicity, at the moment, we assume that the cardinality of the hypothesis function set is finite and is equal to T. We define a gain matrix, $\mathbf{Z}$, where $z_{nt} = y_n h_t(\mathbf{x}_n)$ is the margin of sample $\mathbf{x}_n$ with respect to the t-th hypothesis function h_t. Now let us look at the following minimax optimization problem:

$$\max_{\boldsymbol{\alpha}\in\Gamma(T)} \min_{\mathbf{d}\in\Gamma(N)} \mathbf{d}^T \mathbf{Z} \boldsymbol{\alpha} \ , \tag{5}$$

where $\Gamma(T)$ is the distribution simplex defined as $\Gamma(T) = \{\boldsymbol{\alpha} : \boldsymbol{\alpha} \in \mathcal{R}^T, \sum_{t=1}^T \alpha_t{=}1, \boldsymbol{\alpha}{\geq}0\}$. The above optimization scheme can be understood as finding a set of combination coefficients $\boldsymbol{\alpha}$, such that the performance of the obtained ensemble classifier in the worst case is maximized. It can be shown that this classification scheme is equivalent to the maximum margin classification scheme in Eq. (4):

$$\begin{array}{rl} & \max_{\boldsymbol{\alpha}\in\Gamma(T)} \min_{\mathbf{d}\in\Gamma(N)} \mathbf{d}^T \mathbf{Z} \boldsymbol{\alpha} \ , \\ = & \max_{\boldsymbol{\alpha}\in\Gamma(T)} \min_{1\leq n\leq N} \sum_{t=1}^T \alpha_t z_{nt} \ , \\ = & \max_{\boldsymbol{\alpha}\in\Gamma(T)} \rho \ , \\ & \text{s.t.} \ \sum_{t=1}^T \alpha_t z_{nt} \geq \rho \ , \quad n = 1, \cdots, N \ . \end{array} \tag{6}$$

In the separable data case, a large margin is usually conducive to good generalization in the sense that if a large margin can be achieved with respect to training data, an upper bound on the generalization error is small [26]. However, in noisy data cases where the data distribution is highly overlapped, the optimization scheme of Eq. (5) can be easily misled by outlier samples. Consequently, it will lead to a classifier with a suboptimal performance.

Note that in the minimax problem in Eq. (5), the second minimization problem is optimized over the entire probability space, which is not sufficiently restrictive. A natural strategy is to constrain the distribution or add a penalty term to the objective function to control the distribution skewness, so that the learning algorithm is not allowed to use all of its resources to deal with several hard-to-learn samples. In the following subsections, we will present two regularized boosting algorithms that fall within this framework.

3.1 $\mathbf{LP_{reg}}$-AdaBoost (LPRA)

By constraining the distribution into a box, i.e., $\mathbf{d} \leq \mathbf{c}$, we get the following optimization problem:

$$\max_{\boldsymbol{\alpha}\in\Gamma(T)} \min_{\{\mathbf{d}\in\Gamma(N), \mathbf{d}\leq\mathbf{c}\}} \mathbf{d}^T \mathbf{Z}\boldsymbol{\alpha} \ , \tag{7}$$

where $\mathbf{c}$ is a constant vector, and $\mathbf{d} \leq \mathbf{c}$ means that each element of $\mathbf{d}$ is less than or equal to the corresponding element of $\mathbf{c}$. Eq. (7) can be understood as finding a set of combination coefficients $\boldsymbol{\alpha}$ such that the classification performance in the worst case *within* the distribution box is maximized. After some simple mathematical manipulations, the primal optimization problem of Eq. (7) reads

$$\begin{aligned} \max_{(\rho,\boldsymbol{\lambda},\boldsymbol{\alpha})} \quad & \rho - \sum_{n=1}^{N} c_n \lambda_n \ , \\ \text{s.t.} \quad & \sum_{t=1}^{T} \alpha_t z_{nt} \geq \rho - \lambda_n, \quad n = 1, \cdots, N \ , \\ & \lambda_n \geq 0, \quad n = 1, \cdots, N \ , \\ & \boldsymbol{\alpha} \in \Gamma(T) \ . \end{aligned} \tag{8}$$

$\mathrm{LP_{reg}}$-AdaBoost (LPRA) [17] is a special case of the above optimization scheme in which $c_1 = c_2 = \cdots = c_N = C$. A similar regularization strategy is also used in support vector machine [16] to handle the nonseparable data case. The regularization in Eq. (8) introduces a nonnegative slack variable λ_n into the optimization problem to achieve a soft margin for a pattern:

$$\rho_s(\mathbf{x}_n) = \rho(\mathbf{x}_n) + \lambda_n \ . \tag{9}$$

The relaxation of hard margins allows some patterns to have a smaller margin than ρ, and hence the algorithm does not classify all patterns according to their associated class

labels. The optimization problem of Eq. (8) is usually solved in the dual domain due to implementation issues. It is straightforward to derive the dual problem of Eq. (8) as

$$\begin{aligned} \min_{(\gamma,\mathbf{d})} \quad & \gamma \ , \\ \text{s.t.} \quad & \sum_{n=1}^{N} d_n z_{nt} \leq \gamma, \quad t = 1, \cdots, T \ , \\ & \mathbf{d} \leq \mathbf{c}, \mathbf{d} \in \Gamma(N) \ . \end{aligned} \tag{10}$$

From Eqs. (8) and (10) follows an important observation, that of the dual relationship between achieving a soft margin and controlling the distribution skewness.

For convenience, we reformulate Eq. (7) as

$$\max_{\boldsymbol{\alpha} \in \Gamma(T)} \min_{\mathbf{d} \in \Gamma(N)} \mathbf{d}^T \mathbf{Z} \boldsymbol{\alpha} + \Psi(\|\mathbf{d}\|_\infty) \ , \tag{11}$$

where $\| \cdot \|_p$ is the p-norm and $\Psi(x)$ is a function defined as

$$\Psi(x) = \begin{cases} 0 & , \quad \text{if } x \leq C \ , \\ \infty & , \quad \text{if } x > C \ . \end{cases} \tag{12}$$

Note that the box defined by $\{\mathbf{d} : \|\mathbf{d}\|_\infty \leq C, \mathbf{d} \in \Gamma(N)\}$ is centered on the distribution center $\mathbf{d}_0 = [1/N, \cdots, 1/N]$ (starting point of AdaBoost, see Fig. 1). Also, the parameter C reflects to some extent the distribution skewness between the box boundary and $\mathbf{d}_0$. Eq. (11) indicates that LPRA can be interpreted as a penalty scheme with a penalty of 0 within the box and ∞ outside the box. Therefore, this scheme is somewhat heuristic and may be too restrictive. This analysis suggests that some other smooth penalty functions may be considered for regularization.

Below, we briefly discuss the implementation of LPRA. In real applications, the cardinality of a hypothesis function set, $\mathcal{H}$, can be very large or even infinite. Hence, the gain matrix $\mathbf{Z}$ may not exist in an explicit form, and linear programming cannot be implemented directly. This difficulty can be circumvented by using the column generation (CG) technique [17]. The basic idea of CG is that instead of explicitly solving the optimization problem over the entire set of $\mathcal{H}$, CG restricts the dual problem by only considering the hypothesis functions generated until that moment, and uses the base learner as an "oracle" to generate a new hypothesis, referred to as column, until there are no hypothesis functions within $\mathcal{H}$ that violate the condition $\sum_{n=1}^{N} d_n z_{nt} \leq \gamma$. It has been shown that by using a commercialized linear programming package, the CG based LPRA algorithm can achieve a comparable performance to that of AdaBoost with respect to both classification accuracy and processing time [17]. In the sequel, we will use $|\mathcal{H}|$ to denote the cardinality of a hypothesis function set, and reserve T as the maximum number of iterations of a boosting algorithm.

3.2 LP$_{\mathbf{norm2}}$-AdaBoost (LPNA)

From Eq. (11), which formulates more conveniently the optimization problem of Eq. (7), one plausible strategy to control the skewness of $\mathbf{d}$ is to add a penalty term, $P(\mathbf{d})$, which measures the distances between the query distributions and the distribution center, to the expression in Eq. (5) as follows:

$$\max_{\boldsymbol{\alpha}\in\Gamma(|\mathcal{H}|)} \min_{\mathbf{d}\in\Gamma(N)} \mathbf{d}^T\mathbf{Z}\boldsymbol{\alpha} + \beta P(\mathbf{d}) \ , \tag{13}$$

where β is a user defined parameter that controls the distribution skewness, and hence the training performance.

In this paper, for penalty term $P(\mathbf{d})$, we use the l_2 norm function, which is one of the most popular distance metrics. Since $\mathbf{d}^T\mathbf{Z}\boldsymbol{\alpha} + \beta\|\mathbf{d}-\mathbf{d}_0\|_2$ is convex in $\mathbf{d}$ and concave in $\boldsymbol{\alpha}$, and the sets $\Gamma(N)$ and $\Gamma(T)$ are convex and compact, the max and min operations of Eq. (13) can be interchanged (Generalized Minimax Theorem [27]). Therefore, Eq. (13) can be reformulated as:

$$\max_{\boldsymbol{\alpha}\in\Gamma(|\mathcal{H}|)} \min_{\mathbf{d}\in\Gamma(N)} \mathbf{d}^T\mathbf{Z}\boldsymbol{\alpha}+\beta\|\mathbf{d}-\mathbf{d}_0\|_2 \tag{14}$$

$$= \min_{\mathbf{d}\in\Gamma(N)} \max_{\boldsymbol{\alpha}\in\Gamma(|\mathcal{H}|)} \mathbf{d}^T\mathbf{Z}\boldsymbol{\alpha} + \beta\|\mathbf{d}-\mathbf{d}_0\|_2 \ , \tag{15}$$

$$= \min_{\mathbf{d}\in\Gamma(N)} \max_{1\le j\le|\mathcal{H}|} \textstyle\sum_{n=1}^N d_n z_{nj} + \beta\|\mathbf{d}-\mathbf{d}_0\|_2 \ , \tag{16}$$

$$= \min_{\mathbf{d}\in\Gamma(N)} \gamma + \beta\|\mathbf{d}-\mathbf{d}_0\|_2 \ ,$$

$$\text{s.t. } \textstyle\sum_{n=1}^N d_n z_{nj} \le \gamma, \ j=1,\cdots,|\mathcal{H}| \ , \tag{17}$$

$$= \min_{\mathbf{d}\in\Gamma(N)} \gamma \ ,$$

$$\text{s.t. } \ s_j(\mathbf{d}) = \textstyle\sum_{n=1}^N d_n z_{nj} + \beta\|\mathbf{d}-\mathbf{d}_0\|_2 \le \gamma, \ j{=}1,\cdots,|\mathcal{H}| \ . \tag{18}$$

In the following, we describe how linear programming can be used to solve the optimization problem of Eq. (18). We first define

$$s(\mathbf{d}) = \max_{1\le j\le|\mathcal{H}|} s_j(\mathbf{d}) = \max_{1\le j\le|\mathcal{H}|} \sum_{n=1}^N d_n z_{nj} + \beta\|\mathbf{d}-\mathbf{d}_0\|_2 \ . \tag{19}$$

Since $s_j(\mathbf{d})$ is a convex function in $\mathbf{d}$, $s(\mathbf{d})$ is a piece-wise convex function. Suppose now we have a set of query distributions $\mathcal{S} = \{\mathbf{d}^{(t)}\}_{t=1}^T$. For each query distribution $\mathbf{d}^{(t)}$, we can find one supporting hyperplane for $s(\mathbf{d})$, given by:

$$\gamma = s(\mathbf{d}^{(t)}) + \zeta_t^s(\mathbf{d}-\mathbf{d}^{(t)}) \ , \tag{20}$$

where ζ_t^s is one element of the subdifferential $\partial s(\mathbf{d}^{(t)})$ of s at $\mathbf{d}^{(t)}$. Due to the convexity of s, a supporting hyperplane gives an underestimate of s. More precisely, Eq. (20) can be

expressed as:

$$\begin{aligned}
\gamma &= s(\mathbf{d}^{(t)}) + \zeta_t^s(\mathbf{d} - \mathbf{d}^{(t)}) \ , && (21)\\
&= \max_{1\le j\le|\mathcal{H}|} s_j(\mathbf{d}^{(t)}) + \zeta_t^s(\mathbf{d} - \mathbf{d}^{(t)}) \ , && (22)\\
&= \mathbf{z}_{.t}^T\mathbf{d}^{(t)} + \beta\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2 + \left(\mathbf{z}_{.t} + \beta\frac{\mathbf{d}^{(t)} - \mathbf{d}_0}{\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2}\right)^T(\mathbf{d} - \mathbf{d}^{(t)}) \ , && (23)\\
&= \left(\mathbf{z}_{.t} + \beta\frac{\mathbf{d}^{(t)} - \mathbf{d}_0}{\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2}\right)^T\mathbf{d} \ , && (24)
\end{aligned}$$

where $\mathbf{z}_{.t}$ is the t-th column of matrix $\mathbf{Z}$ given by

$$\mathbf{z}_{.t} = [y_1 h_t(\mathbf{x}_1), \cdots, y_N h_t(\mathbf{x}_N)]^T \ , \tag{25}$$

and

$$h_t = \arg\max_{h\in\mathcal{H}} \sum_{n=1}^{N} d_n^{(t)} h(\mathbf{x}_n) y_n \ . \tag{26}$$

Let us define

$$\widetilde{\mathbf{Z}} = \mathbf{Z} + \beta\left[\frac{\mathbf{d}^{(1)} - \mathbf{d}_0}{\|\mathbf{d}^{(1)} - \mathbf{d}_0\|_2}, \cdots, \frac{\mathbf{d}^{(T)} - \mathbf{d}_0}{\|\mathbf{d}^{(T)} - \mathbf{d}_0\|_2}\right] \ . \tag{27}$$

Analogous to $\mathbf{z}_{.t}$, the columns of $\widetilde{\mathbf{Z}}$ are denoted as $\tilde{\mathbf{z}}_{.t}$.

It follows from Eq. (24) that the optimization problem of Eq. (18) can be linearly approximated as:

$$\begin{aligned}
\min_{(\gamma,\mathbf{d})} \quad & \gamma \ , \\
\text{s.t.} \quad & \tilde{\mathbf{z}}_{.t}^T\mathbf{d} \le \gamma, \quad t = 1, \cdots, T \ , \\
& \mathbf{d} \in \Gamma(N) \ ,
\end{aligned} \tag{28}$$

which is much easier to deal with than the original problem of Eq. (18). Fig. 2 illustrates the proposed linear approximation of γ. Note that this is only a linear approximation, which in general becomes better as more constraints are added.

The query distributions can be obtained by using the well-known column generation technique. The column generation guarantees convergence of the optimization problem of Eq. (18) in a finite number of iteration steps. The same technique is used for LP_{reg}-AdaBoost in [17]. Due to the degeneracy of Eq. (28), the column generation, however, usually shows a pattern of slow convergence. Note that the problem of slow convergence, particularly in the initial several iterations, is due to the spareness of the optimum solution produced in each iteration. Not only does this make the learning process difficult, but this also produces many unnecessary columns. The degeneracy problem is illustrated in Fig. 2. The circled numbers in Fig. 2 enumerate the generated columns (constraints). Note

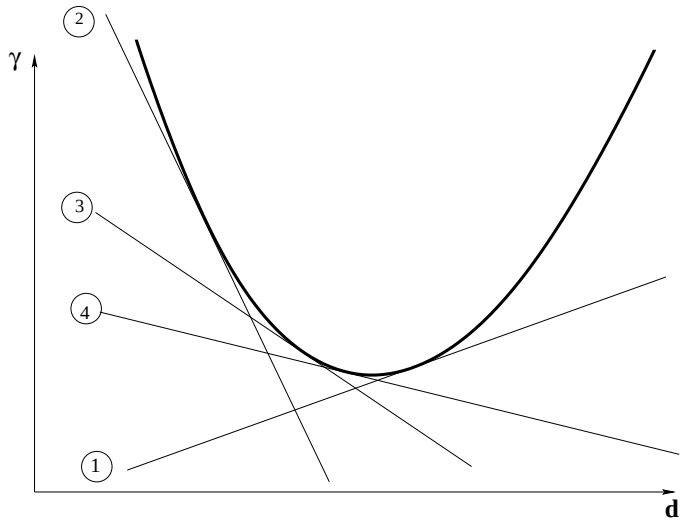


Figure 2: The slow convergence problem of column generation. The numbers in the circle are the sequences of generated columns or constraints. It is clear that, given the columns of 1 and 4, the columns of 2 and 3 will not be activated, i.e., the corresponding hypothesis coefficients α_2 and α_3 are equal to zero.

that in the example of Fig. 2 given the columns of 1 and 4, the columns of 2 and 3 will not be activated. This means that the corresponding hypothesis coefficients α_2 and α_3 equal to zero, according to the Karush-Kuhn-Tucker (KKT) condition [28]. Therefore, the generation of h_2 and h_3 is redundant, as they do not contribute to the final solution.

One natural idea to alleviate the slow convergence problem is to constrain the solution within a box, centered at the previous solution, also called the BOXSTEP method [29]:

$$\begin{aligned} \min_{(\gamma,\mathbf{d})} \quad & \gamma \ , \\ \text{s.t.} \quad & \tilde{\mathbf{z}}_{.t}^T \mathbf{d} \leq \gamma, \quad t = 1, \cdots, T \ , \\ & \mathbf{d} \in \Gamma(N), \|\mathbf{d} - \mathbf{d}^{(T)}\|_\infty \leq B \ , \end{aligned} \tag{29}$$

where the parameter B defines the box size. Note that:

$$\begin{aligned} \|\mathbf{d} - \mathbf{d}^{(T)}\|_\infty = \quad & \max_{1 \leq n \leq N} |d(n) - d^{(T)}(n)| \leq B \Rightarrow \\ & \mathbf{d}^{(T)} - \mathbf{1}B \leq \mathbf{d} \leq \mathbf{1}B + \mathbf{d}^{(T)} \ , \end{aligned} \tag{30}$$

which together with the constraint of $\mathbf{d} \in \Gamma(N)$ gives:

$$\max\{\mathbf{d}^{(T)} - \mathbf{1}B, \mathbf{0}\} \leq \mathbf{d} \leq \mathbf{1}B + \mathbf{d}^{(T)} \ . \tag{31}$$

Consequently, the optimization problem in Eq. (29) can be further simplified as the following

$\mathbf{LP_{norm2}}$-AdaBoost

Initialization: $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, the maximum number of iterations T, parameter β, Box size B, data distribution $d^{(1)}(n) = 1/N$, for $n = 1 : N$,

for $t = 1 : T$

1. Train base learner with respect to distribution $\mathbf{d}^{(t)}$ and get hypothesis $h_t(\mathbf{x}) : \mathbf{x} \to \{\pm 1\}$.

2. Solve the optimization problem:

$$\begin{aligned}(\mathbf{d}^*, \gamma^*) = \quad & \arg\min_{(\gamma, \mathbf{d})} \quad \gamma \ , \\ & \text{s.t.} \qquad \tilde{\mathbf{z}}_{\cdot j}^T \mathbf{d} \le \gamma, \quad j = 1, \cdots, t \ , \\ & \qquad\quad \max\{\mathbf{d}^{(t)} - \mathbf{1}B, \mathbf{0}\} \le \mathbf{d} \le \mathbf{1}B + \mathbf{d}^{(t)} \ , \\ & \qquad\quad \textstyle\sum_{n=1}^N d_n = 1 \ .\end{aligned}$$

3. Update weights as $\mathbf{d}^{(t+1)} = \mathbf{d}^*$.

end

Output : $F(\mathbf{x}) = \sum_{t=1}^T \alpha_t^* h_t(\mathbf{x})$

where $\boldsymbol{\alpha}^*$ is the Lagrangian multipliers from the last LP.

Figure 3: Pseudo-code of $\mathrm{LP}_{\mathrm{norm2}}$-AdaBoost algorithm.

LP problem:

$$\begin{aligned}\min_{(\gamma, \mathbf{d})} \quad & \gamma \ , \\ \text{s.t.} \quad & \tilde{\mathbf{z}}_{\cdot t}^T \mathbf{d} \le \gamma, \quad t = 1, \cdots, T \ , \\ & \max\{\mathbf{d}^{(T)} - \mathbf{1}B, \mathbf{0}\} \le \mathbf{d} \le \mathbf{1}B + \mathbf{d}^{(T)} \ , \\ & \sum_{n=1}^N d_n = 1 \ .\end{aligned} \tag{32}$$

Eq. (32) gives rise to a new LP based boosting algorithm, which we refer to as $\mathrm{LP}_{\mathrm{norm2}}$-AdaBoost (LPNA), summarized in Fig. 3.

The proposed algorithm may be better understood in the primal domain. The dual

form of Eq. (28) is:

$$\begin{aligned} \max_{(\rho,\boldsymbol{\alpha})} \quad & \rho \\ \text{s.t.} \quad & \sum_{t=1}^{T} \alpha_t z_{nt} + \beta \sum_{t=1}^{T} \alpha_t \frac{d_n^{(t)} - 1/N}{\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2} \geq \rho, \; n = 1, \cdots, N \; , \\ & \boldsymbol{\alpha} \in \Gamma(T) \; . \end{aligned} \tag{33}$$

Similar to Eq. (8), Eq. (33) leads to the following definition of a sample soft margin:

$$\rho_s(\mathbf{x}_n) = \sum_{t=1}^{T} \alpha_t z_{nt} + \beta \sum_{t=1}^{T} \alpha_t \frac{d_n^{(t)} - 1/N}{\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2} \tag{34}$$

where the term $(\beta \sum_{t=1}^{T} \alpha_t \frac{d_n^{(t)} - 1/N}{\|\mathbf{d}^{(t)} - \mathbf{d}_0\|_2})$ can be interpreted as a "mistrust" in examples. Here, the rationale is that a pattern which is often visited by the base learner (i.e., hard to classify correctly) will have a high average distribution, and should have less influence on the outcome of the final classifier. The parameter β controls the tradeoff between margin and "mistrust". It is interesting to note that our soft margin definition given by Eq. (34) is very similar to that of AdaBoost$_{\text{Reg}}$, defined as:

$$\rho_{\text{Reg}}(\mathbf{x}_n) = \sum_{t=1}^{T} \alpha_t z_{nt} + \beta \sum_{t=1}^{T} \alpha_t d_n^{(t)} \; , \tag{35}$$

which is introduced in AdaBoost on the algorithm level [10]. The main difference is that our soft margin is calculated with respect to the center distribution. If the query distributions of a pattern, say $\mathbf{x}_n$, are always less than $1/N$ (i.e., $\mathbf{x}_n$ is an easy example), the "mistrust" in Eq. (34) can take a negative value. It means that the soft margin penalizes some hard examples and at the same time rewards some easy examples. In [10, 23], it was experimentally observed that AdaBoost increases the margin of the most hard-to-learn examples at the cost of reducing the margins of the rest of the data. Therefore, by defining a soft margin as in Eq. (34), we seek to reverse the boosting process to some extent, the strength of which is controlled by β.

Below, we briefly discuss the implementation of the proposed LPNA. One important parameter of LPNA is the box size, B. If B is too large, the algorithm will still suffer from a slow convergence problem, and if B is too small, the updating of query distributions may not be adequate, affecting the convergence rate of the algorithm. From our experience, we find that $B \in [\frac{3}{N}, \frac{10}{N}]$ is appropriate, and in our experiments we choose $B = \frac{5}{N}$.

4 Base Learner: RBF Network

In this paper, as the base learner of a specific boosting algorithm, we use the radial basis function (RBF) network [30]. The RBF network is a multi-dimensional nonlinear mapping

based on the distances between the input vector and predefined *center vectors*. Using the same notation as in Section 2, the mapping is specified as a weighted combination of J basis functions:

$$h(\mathbf{x}) = \sum_{j=1}^{J} \pi_j \phi_j(\|\mathbf{x} - \mathbf{c}_j\|_p), \tag{36}$$

where $\phi_j(\|\mathbf{x} - \mathbf{c}_j\|_p)$ is a radial basis function, π_j is a weight parameter, and J is a predefined number of RBF centers, $\mathbf{c}_j$, $j = 1, \cdots, J$. Also, basis functions $\phi_j(\cdot)$ are arbitrary nonlinear functions, and $\|\cdot\|_p$ denotes the p-norm (usually assumed Euclidean).

In the literature, one of the most popular RBF nets is the Gaussian RBF network [31], where the basis functions are specified as the unnormalized form of the Gaussian density function given by

$$g(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right), \tag{37}$$

where $\boldsymbol{\mu}$ is the mean and $\boldsymbol{\Sigma}$ is the covariance matrix. For simplicity, $\boldsymbol{\Sigma}$ is often assumed to have the form $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Hence, the Gaussian RBF network is given by

$$h(\mathbf{x}) = \sum_{j=1}^{J} \pi_j g_j(\mathbf{x}) = \sum_{j=1}^{J} \pi_j \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j\|_2^2}{2\sigma_j^2}\right), \tag{38}$$

where the $\boldsymbol{\mu}$'s represent center vectors, and the σ's can be interpreted as the width of basis functions.

The parameters of the Gaussian RBF network – namely, the means $\{\boldsymbol{\mu}_j\}_j$, the variances $\{\sigma_j^2\}_j$, and the weighting parameters $\{\pi_j\}_j$ – are learned on training samples. In this paper, we employ an iterative learning algorithm, where all the RBF parameters are simultaneously computed by minimizing the following error function [10]:

$$E = \frac{1}{2} \sum_{n=1}^{N} (y_n - h(\mathbf{x}_n))^2 + \frac{\lambda}{2N} \sum_{j=1}^{J} \pi_j^2 \,, \tag{39}$$

where λ is a regularization constant. In the first step, the means $\{\boldsymbol{\mu}_j\}_j$ are initialized by the standard K-means clustering algorithm, while the variances $\{\sigma_j\}_j$ are determined as the distance between $\boldsymbol{\mu}_j$ and the closest $\boldsymbol{\mu}_i$, $(i \neq j,\ i \in [1, J])$. Then, in the following iteration steps, a gradient descent of the error function in Eq. (39) is performed to update $\{\boldsymbol{\mu}_j\}_j$, $\{\sigma_j^2\}_j$, and $\{\pi_j\}_j$. In this manner, the network fine-tunes itself to training data.

5 Experimental Results

To demonstrate the effectiveness of the proposed algorithm, a large-scale experiment is conducted. The performance of our algorithm is compared with that of the following

Table 1: Summary of 12 datasets

	No. Features	No. Train	No. Test	No. Realizations
Banana	2	400	4900	100
Bcancer	9	200	77	100
Diabetics	8	468	300	100
German	20	700	300	100
Heart	13	170	100	100
Ringnorm	20	400	7000	100
Fsolar	9	666	400	100
Thyroid	5	140	75	100
Titanic	3	150	2051	100
Waveform	21	400	4600	100
Splice	60	1000	2175	20
Twonorm	20	400	700	100

classifiers: (1) the radial basis function (RBF) network, described in Sec. 4, (2) AdaBoost, discussed in Sec. 2, (3) AdaBoost$_{\text{Reg}}$, mentioned in Sec. 1 and detailed in [2, 10, 13], and (4) LPRA, explained in Sec. 3.1 and detailed in [17]. The radial basis function net is used as the base learner for all boosting algorithms. All of the RBF parameters, including the number of the RBF centers and the number of training iterations, are the same as those used in [10]. Here, we avoid verbatim repetition of the detailed, lengthy description of the RBF parameters, and refer the reader to [10].

We use 12 artificial and real-world datasets: *banana, waveform, diabetics, breast cancer, ringnorm, twonorm, splice, heart, german, titanic, thyroid and flare solar.* The datasets are taken from the publicly available UCI, DELVE and STATLOG benchmark repositions. Alterations to these datasets can be obtained from the IDA repository [32], where the datasets have been randomly partitioned into 100 realizations of training and testing data (*splice* has only 20 realizations). The detailed data information is summarized in Table 1.

In the experiments, the tuning of the regularization parameter β is done through cross-validation based on training data. Throughout, the maximum number of training iterations of LPNA and LPRA is heuristically set to $T = 150$. As typically done in the literature [33, 34], we preset the number of training steps to obtain a reasonable comparison of the performance of the algorithms balanced against certain processing-time constraints. The commercialized optimization package XPRESS is used as the LP solver.

In the sequel, we present several examples in which we illustrate the properties of the proposed algorithm. First, we show classification results on *banana* data set, whose samples are characterized by two-dimensional feature vectors. The decision boundaries of RBF, AdaBoost and LPNA are plotted in Fig. 4. Note that AdaBoost tries to classify each pattern according to its associated class label, and forms a "zigzag" decision boundary,

which clearly illustrates the overfitting phenomenon of AdaBoost. Both RBF and LPNA yield smooth and similar decision boundaries. In this case, it is difficult to determine which decision boundary is better visually. This indicates that RBF with well tuned structural parameters is a strong classifier. Hence, it is not surprising that boosting such a strong classifier without any regularization will easily lead to overfitting.

In the second example, we present the training and testing results, and margin plots of AdaBoost and LPNA based on one realization of *waveform* data in Fig. 5. AdaBoost tries to maximize the margin of each pattern, and hence effectively reduces the training error to zero. However, it quickly leads to overfitting. In contrast, LPNA tries to maximize the soft margin, purposely allowing some difficult-to-learn examples to remain with small hard margins. Thereby, LPNA effectively alleviates the overfitting problem of AdaBoost. As can be seen in Fig. 5, in the beginning the classification error for LPNA on test data gradually decreases as the number of iterations become larger. After a certain number of iterations, the classification error for LPNA reaches a stable minimum value. Similar performance of LPNA is observed over all the datasets used.

Table 2 presents a more comprehensive comparison of LPNA with the following algorithms: RBF [10], AdaBoost [10], AdaBoost$_{\text{Reg}}$ [10], and LP$_{\text{reg}}$-AdaBoost (LPRA) [17]. The comparison concerns the average classification results and their standard deviations over 100 realizations of the 12 datasets, as detailed in Table 2. The best results are marked in boldface. From the table, we note the following:

1. AdaBoost with RBF as a base learner performs worse than a single RBF classifier in almost all cases. This is due to overfitting of AdaBoost. In many cases, AdaBoost leads to overfitting quickly, only after a few iterations, which clearly indicates that regularization is needed for AdaBoost.

2. LPNA can significantly improve the performance of RBF. Moreover, the regularization of LPNA alleviates the overfitting problem of AdaBoost.

3. We observe that LPNA achieves a similar, and in some cases (e.g., *waveform*) better classification performance than AdaBoost$_{\text{Reg}}$. This highlights the success of our approach. Note that AdaBoost$_{\text{Reg}}$ has been established as one of the best regularized AdaBoost algorithms, which reportedly outperforms support vector machine with RBF kernel on the given 12 datasets [10].

4. We also make a comparison between LPRA and LPNA. Though LPRA can achieve better performance than AdaBoost, in almost all cases LPRA is inferior to LPNA. This may be explained due to a heuristic hard-limited penalty function used in LPRA that is inferior to the penalty function used in LPNA.

Table 2: Classification errors and standard deviations (%) of the RBF, AdaBoost(AB), $\text{AdaBoost}_{\text{Reg}}(AB_R)$, LP_{norm2}-AdaBoost (LPNA), LP_{reg}-AdaBoost (LPRA) [17]. The best results are marked in boldface.

	RBF [10]	AB [10]	AB_R [10]	LPNA	LPRA
Banana	10.8±0.6	12.3±0.7	10.9±0.4	**10.7±0.4**	10.9±0.9
Waveform	10.7±1.1	10.8±0.6	9.8±0.8	**9.4±0.4**	9.8±0.5
Bcancer	27.6±4.7	30.4±4.7	26.5±4.5	**25.9±4.5**	26.7±4.7
Diabetics	24.3±1.9	26.5±2.3	**23.8±1.8**	**23.8±1.8**	24.3±2.0
German	24.7±2.4	27.5±2.5	24.3±2.1	**23.9±2.3**	24.5±2.3
Heart	17.6±3.3	20.3±3.4	**16.5±3.5**	16.9±3.2	17.5±3.6
Ringnorm	1.7±0.2	1.9±0.3	**1.6±0.1**	**1.6±0.2**	1.7±0.2
Fsolar	34.4±2.0	35.7±1.8	**34.2±2.2**	34.3±1.8	34.6±2.0
Thyroid	4.5±2.1	4.4±2.2	4.6±2.2	**4.3±2.2**	4.4±2.1
Titanic	23.3±1.3	22.6±1.2	22.6±1.2	**22.5±1.1**	23.3±0.9
Splice	10.0±1.0	10.1±0.5	9.5±0.7	**9.4±0.7**	9.5±0.6
Twonorm	2.9±0.3	3.0±0.3	**2.7±0.2**	**2.7±0.2**	2.8±0.2

5. The proposed LPNA algorithm can be readily extended to the case in which the base learner produces a soft decision. The soft decisions of RBF can be computed as: $h(\mathbf{x}) = (\mathrm{e}^{2z} - 1)/(\mathrm{e}^{2z} + 1) \in [-1, +1]$, where $z \in \mathcal{R}$ is the output of RBF given a pattern $\mathbf{x}$. The classification results of LPNA(S) that uses the outlined soft information of RBF are reported in Table 3. By exploiting the confidence information, LPNA(S) performs slightly better than the original LPNA that uses hard RBF decisions (LPNA(H)). In machine learning the notion of convergence is associated to the number of iteration steps it takes an algorithm to achieve a specific classification error. As can be seen in Fig. 6, for LPNA(H) it takes 150 itertations, while for LPNA(S), only 50 iterations to achieve the same classification error. Typically, the experiments show that LPNA(S) converges faster than LPNA(H) with respect to the classification error on the test datasets.

Finally, note that complexity of LPNA is equal to that of the column-generation based LPRA algorithm [17]. It has been shown that by using a commercialized linear programming package, the column-generation based LPRA algorithm can achieve a comparable performance to that of AdaBoost with respect to both classification accuracy and processing time [17].

Table 3: Classification errors and standard deviations of LPNA(H) and LPNA(S) that use hard and soft decisions, respectively.

	LPNA(H)	LPNA(S)
Banana	10.7±0.4	**10.6±0.4**
Waveform	9.4±0.4	**9.3±0.4**
Bcancer	**25.9±4.5**	26.5±4.6
Diabetis	23.8±1.8	23.8±1.8
German	23.9±2.3	24.0±2.2
Heart	16.9±3.2	17.0±3.1
Ringnorm	1.6±0.2	1.6±0.2
Fsolar	34.3±1.8	34.2±2.0
Thyroid	4.3±2.2	**4.2±2.2**
Titanic	22.5±1.1	22.5±1.1
Splice	9.4±0.7	**9.3±0.7**
Twonorm	2.7±0.2	2.7±0.2

6 Conclusions

In this paper, we have addressed the problem of overfitting in AdaBoost in noisy settings, which may hinder the implementation of AdaBoost for real-world applications. By exploring a close connection between AdaBoost and linear programming, we have proposed a new regularized AdaBoost algorithm – LP_{norm2}–AdaBoost, or short LPNA. The algorithm is based on an intuitive idea of controlling the data distribution skewness in the learning process by introducing a smooth convex penalty function into the objective of the minimax problem. Thereby, outliers are prevented from spoiling decision boundaries in training. We have used the stabilized column generation technique to transform the optimization problem into a simple linear programming problem.

We have presented the results of a large-scale experiment on 12 datasets, in which LPNA is also compared with the following classifiers: RBF net [30], AdaBoost [1], $\text{AdaBoost}_{\text{Reg}}$ [10], and LPRA [17]. Empirical results show that LPNA effectively alleviates the overfitting problem of AdaBoost, and achieves a slightly better overall classification performance than to date the best regularized AdaBoost algorithm called $\text{AdaBoost}_{\text{Reg}}$.

The empirical validation clearly indicates the success of the proposed approach, especially in light of years' long dominance of $\text{AdaBoost}_{\text{Reg}}$. More importantly, unlike $\text{AdaBoost}_{\text{Reg}}$, where regularization is heuristically introduced on the algorithm level, our LPNA has a clear underlying optimization scheme.

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[2] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," in *Advanced Lectures on Machine Learning*, S. Mendelson and A. Smola, Eds. Springer, 2003, pp. 119–184.

[3] H. Schwenk, "Using boosting to improve a hybrid HMM/Neural Network speech recognizer," in *Proc. Intl. Conf. Acoustics, Speech, Signal Processing*, Phoenix, AZ, USA, 1999, pp. 1009–1012.

[4] R. Zhang and . I. Rudnicky, "Improving the performance of an LVCSR system through ensembles of acoustic models," in *Proc. Intl. Conf. Acoustics, Speech, Signal Processing*, vol. 1, Hong Kong, 2003, pp. 876–879.

[5] G.Tur, R. E. Schapire, and D. Hakkani-Tur, "Active learning for spoken language understanding," in *Proc. of IEEE Int. Conf. on Acoustics, Speech and Signal Proc.*, Hong Kong, China, 2003.

[6] J. Miteran, J. Matas, E. Bourennane, M. Paindavoine, and J. Dubois, "Automatic hardware implementation tool for a discrete adaboost-based decision algorithm," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 7, pp. 1035–1046, 2005.

[7] R. Nishii and S. Eguchi, "Robust supervised image classifiers by spatial adaboost based on robust loss functions," in *Proc. SPIE, Image and Signal Processing for Remote Sensing XI*, vol. 5982, no. 1, 2005.

[8] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kegl, "Meta-features and AdaBoost for music classification," *in press, Machine Learning*, 2006.

[9] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.

[10] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, 2001.

[11] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] A. J. Grove and D. Schuurmans, "Boosting in the limit: maximizing the margin of learned ensembles," in *Proc. 15th Nat'l Conf. on Artificial Intelligence*, Madison, WI, USA, 1998, pp. 692–699.

[13] G. Rätsch, "Robust boosting via convex optimization: theory and application," Ph.D. dissertation, University of Potsdam, Germany, 2001.

[14] L. Breiman, "Prediction games and arcing algorithms," *Neural Computation*, vol. 11, no. 7, pp. 1493–1517, October 1999.

[15] Y. Freund and R. E. Schapire, "Game theory, on-line prediction and boostin," in *Proc. 9th Annual Conf. Computational Learning Theory*, Desenzano del Garda, Italy, 1996, pp. 325–332.

[16] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[17] A. Demiriz, K. P. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *Machine Learning*, vol. 46, pp. 225–254, 2002.

[18] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[19] ——, "Arcing classifiers," *The Annals of Statistics*, vol. 26, no. 3, pp. 801–849, 1998.

[20] L. Mason, J. Bartlett, P. Baxter, and M. Frean, "Functional gradient techniques for combining hypotheses," in *Advances in Large Margin Classifiers*, B. Scholkopf, A. Smola, P. Bartlett, and D. Schuurmans, Eds. Cambridge, MA, USA: MIT Press, 2000, pp. 221–247.

[21] W. Jiang, "Some theoretical aspects of boosting in the presence of noisy data," in *Proc. 18th Intl. Conf. on Machine Learning*, Williams College, MA, 2001, pp. 234–241.

[22] ——, "Is regularization unnecessary for boosting," in *Proc. Eighth Intl. Workshop on Artificial Intelligence and Statistics*, Key West, FL, 2001.

[23] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," in *Proc. 14th Intl. Conf. on Machine Learning*, Nashville, TN, USA, 1997, pp. 322–330.

[24] C. Rudin, I. Daubechies, and R. E. Schapire, "The dynamics of AdaBoost: Cyclic behavior and convergence of margins," *J. Machine Learning Research*, vol. 5, pp. 1557–1595, Dec 2004.

[25] J. von Neumann, "Zur theorie der gesellschaftsspiele," *Mathematische Annalen.*, vol. 100, pp. 295–320, 1928.

[26] V. Vapnik, *Statistical Learning Theory.* New York: Wiley, 1998.

[27] I. Ekeland and R. Temam, *Convex Analysis and Variational Problems.* Amsterdam, Holland: North-Holland Pub. Co., 1976.

[28] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization.* New York: John Wiley and Sons Inc., 2001.

[29] R. E. Marsten, W. W. Hogan, and J. W. Blankenship, "The BOXSTEP method for large-scale optimization," *Operations Research*, vol. 23, pp. 389–405, 1975.

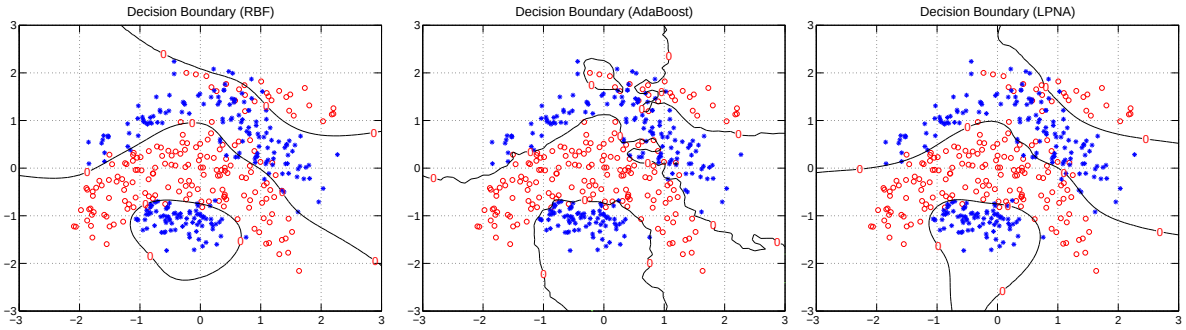
[30] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.

[31] C. Bishop, *Neural Networks for Pattern Recognition.* Oxford: Claredon Press, 1995.

[32] G. Rätsch, "IDA benchmark repository," 2001. [Online]. Available: http://ida.first.fhg.de/projects/bench/benchmarks.htm

[33] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, 1999.

[34] R. E. Schapire, "Using output codes to boost multiclass learning problems," in *Proc. 14th Intl. Conf. Machine Learning*, Nashville, TN, USA, 1997, pp. 313–321.



(a) RBF (b) AdaBoost (c) LPNA

Figure 4: The decision boundaries of three methods: RBF, AdaBoost and LPNA based on one realization of the banana data. AdaBoost tries to classify each pattern according to its associated label and forms a zigzag decision boundary, which gives a straightforward illustration of the overfitting phenomenon of AdaBoost. Both RBF and LPNA give smooth and similar decision boundaries in this case.

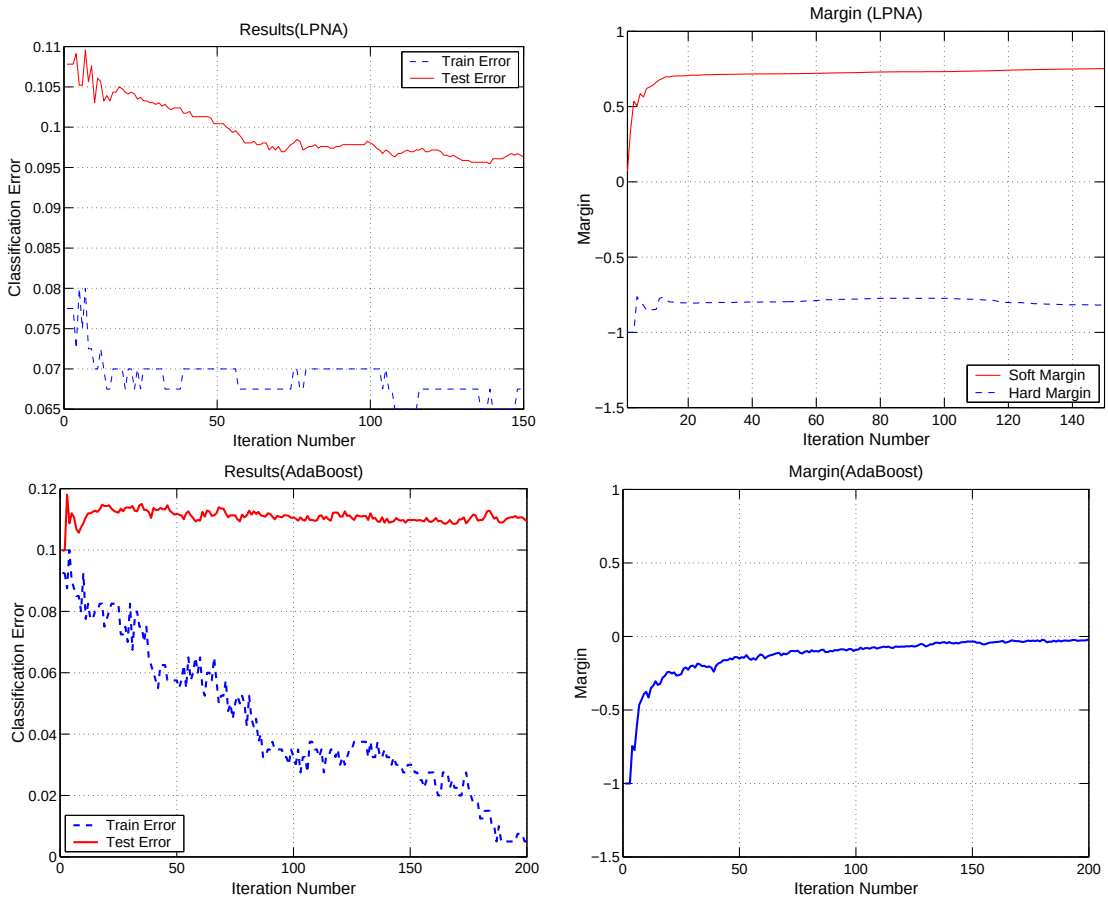


Figure 5: Training and testing results, and margin plots of two methods: AdaBoost and LPNA based on one realization of *waveform* data. AdaBoost can effectively reduce the training error to zero but leads to overfitting only after a few iterations. LPNA effectively alleviates the overfitting problem.

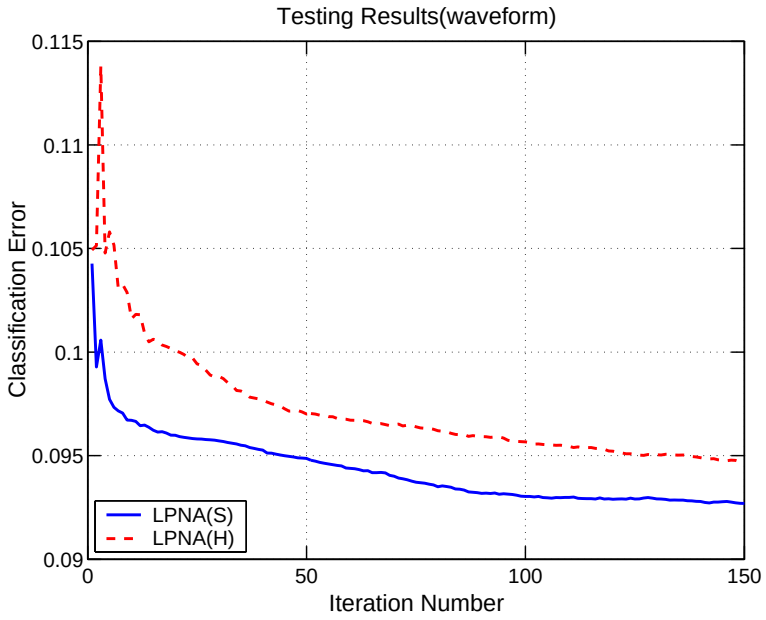


Figure 6: Testing results of LPNA(H) and LPNA(S) averaged over 100 realizations of *waveform* dataset. LPNA(S) converges faster than LPNA(H) in terms of the classification performance.