Nonlinear Boost

Chu-Song Chen*, Chang-Ming Tsai, Jiun-Hung Chen, and Chia-Ping Chen

Institute of Information Science, Academia Sinica, Nankang 115, Taipei, Taiwan.

Email: song@iis.sinica.edu.tw

TEL: +886-2-27883799 ext 1310 FAX: +886-2-27824814

* The corresponding author

Abstract

In this paper, we propose the post-classification scheme that is useful for improving weak-hypothesis combination of AdaBoost. The post-classification scheme allows the weak hypotheses to be combined nonlinearly, and can be shown to have a generally better performance than the original linear-combination approach in either theory or practice. The post-classification scheme provides a general perspective on combining weak hypotheses, in which many existing boosting methods can be treated as its sub-cases. By using support vector machine (SVM) as post-classifier, we have shown that AdaBoost and SVM can be effectively combined in the post-classification scheme to achieve an effective classification algorithm. Experiment results show that the proposed nonlinear boost approach can significantly improve the performance of AdaBoost.

Index Terms: Classification, boost, AdaBoost, nonlinear boost, SVM, neural networks.

I. Introduction

Given a set of input-output training pairs for a binary classification problem, $S \equiv \{(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N) \mid x_i \in R^d, y_i \in \{1,-1\}, i = 1, \ldots, N\}$, the classification framework of boosting [22][27] combines simple member classifiers with appropriate weights to form an ensemble, which improves the performance of each ensemble member. Let $\{h_1, h_2, \ldots, h_T \mid h_i : R^d \rightarrow \{1,-1\}\}$ be a set of hypotheses. Consider the composite ensemble hypothesis

$$f(x) = \sum_{t=1}^{T} a_t h_t(x), \tag{1}$$

where coefficients $a_1, a_2, \ldots,$ and a_T are nonnegative. Both the member classifiers h_t and the coefficients a_t ($t = 1, \ldots, T$) can be found by using the boosting procedure. The idea of boosting stems from the observation that finding many rough or weak rules is much easier than finding a single, highly accurate rule.

Theoretically, by investigating Valiant's probably approximately correct (PAC) learning, Kearns and Valiant [15] prove that weak classifiers, which perform only slightly better than random, are possible to be combined to form an arbitrarily good ensemble classifier (when enough data is available). AdaBoost [9][27], which is the first step toward more practical boosting algorithms [22], provides a provably correct procedure that allows classifier designers to continue adding weak learners until a desired low training error is achieved.

AdaBoost has been used in a variety of applications such as face detection [2][19][30], face recognition [10], gesture recognition [21], information retrieval [14][26] and speech recognition [2][31]. In [30], AdaBoost is employed to construct a cascade of weak classifiers called rectangle feature classifiers for fast face detection. In [21], AdaBoost using exemplar-based classifiers was introduced for online retraining to improve real-time

gesture recognition. In [19], a multi-view face detection system was created by FloatBoost, which integrated Adaboost with a feature selection method to cull insignificant weak learners. In [2], a structured learning algorithm, Max-Select AdaBoost, was introduced for speaker detection. In [10], AdaBoost served as a binary classification tool, and was combined with a multi-class comparison technique called constrained majority voting to deal with multi-class problems for face recognition.

AdaBoost aims to maintain a distribution or set of weights over a training set. Initially, weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that a weak (or base) learner is forced to focus on the hard examples in the training set. The iterative re-weighting procedure allows AdaBoost to generate a set of classifiers. Each classifier can be thought of as a professional who is particularly suitable for classifying the re-weighted (or re-distributed) data set. In AdaBoost and many of its variations [5][9][19][24][28], these classifiers are combined linearly as shown in (1), where each classifier is assigned a fixed weight.

In this paper, we consider a new ensemble scheme in which the classifiers trained in each round can be combined nonlinearly. The motivations for using a nonlinear-ensemble framework are:

(1) The classifier generated in each round of AdaBoost is treated as an expert in classifying the training data set sampled with a particular distribution, and the distribution peaks at some data vectors those are badly (or difficult to be) classified in the last round. These classifiers are thus better to be combined adaptive to the input vector or the associated output label, so that the ensemble classifier assigns a high weight for a classifier when the input vector is close to some of the distribution peaks in association with this classifier. Note that this ensemble scheme yields distinct weights for distinct input vectors or

classification outputs, and is generally nonlinear for classifier combination.

(2) Note that focusing only on training errors of badly classified data may yield classifiers with insufficient generalization ability. Researches focusing on increasing the generalization ability of AdaBoost [24][28] use soft margins to reduce over-fitting effects. Based on a nonlinear ensemble scheme, we will show that regularization frameworks can be built more generally along with AdaBoost so that the output classifier can earn high generalization ability.

In this paper, the idea of *post-classification* (referred to as "classification after classification") is proposed to generate nonlinear ensembles of the classifiers generated in AdaBoost rounds. We will discuss ways that appropriate post-classifiers can be designed so that the resultant classifier performs better than the one obtained using the original linear combination approach. The new method proposed in this paper is called *nonlinear boost*. Distinguishing features of nonlinear boost include allowing an additional bias term in a linear ensemble, allowing nonlinear ensemble, and guaranteeing that the training performance of the resulting classifier is no worse than that of each weak hypothesis.

The rest of this paper is organized as follows. Section II introduces the post-classification scheme. Section III presents our approach. Section IV gives experiment results. Section V presents some further remarks. Conclusions and future works are given in Section VI.

II. Post-classification Scheme

A. Review of AdaBoost

In AdaBoost, a non-negative weight vector $\mathbf{w}^{(t)} = (w_1^{(t)}, w_2^{(t)}, \ldots, w_N^{(t)})$ is assigned to data at

step t, and a weak learner h_t is constructed based on $\mathbf{w}^{(t)}$, as shown below.

Algorithm *AdaBoost* [9][27]: (Assume that a weak learner has been given.)

1. **Input**: S={(x_1,y_1), (x_2,y_2), …, (x_N,y_N)}. Set number of iterations, T.
2. **Initialize**: $w_n^{(1)} = 1/N$, for all n=1, …, N.
3. **Do for** t = 1, …, T,
   (a) Use the weak learner to train a classifier with respect to the weighted sample set {S,$\mathbf{w}^{(t)}$} and obtain a hypothesis h_t: $\boldsymbol{R}^d \to \{-1,+1\}$.
   (b) Calculate the weighted training error ε_t of h_t:

$$\varepsilon_t = \sum_{n=1}^{N} w_n^{(t)} I\left(y_n \neq h_t(x_n)\right), \tag{2}$$

   where I(statement) = 1 if the statement is true; otherwise, I(statement) = 0.
   (c) **If** ε_t = 0 or ε_t > 1/2, **then** set $T = t-1$ and **goto** Step 4.
   (d) Set

$$a_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}. \tag{3}$$

   (e) Update weights

$$w_n^{(t+1)} = \frac{1}{Z_t} w_n^{(t)} \exp\{(-a_t y_n h_t(x_n))\}, \tag{4}$$

   where Z_t is a normalization constant, such that $\sum_{n=1}^{N} w_n^{(t+1)} = 1$.
4. **Output**:

$$f_T(x) = \sum_{t=1}^{T} a_t h_t(x). \tag{5}$$

In the algorithm, the weight vector is updated at each iteration t according to the weighted error incurred by the weak learner in the last iteration. In this way, misclassified examples get higher weights and the rest get lower weights in the next iteration. In essence, AdaBoost is a procedure for finding a mapping f which attempts to lower the following value:

$$\sum_n \boldsymbol{exp}\{-y_n f(x_n)\}, \tag{6}$$

$$= \sum_n \boldsymbol{exp}\left\{-y_n \sum_t a_t h_t(x_n)\right\}. \tag{7}$$

The AdaBoost algorithm ensures that, when the number of iterations T increases, the error measure defined in (6) and (7) decreases if every weak classifier performs better than random guess, i.e., ε_t (defined in (2)) is larger than $0.5+\eta$ for all t, where η is a positive constant. Note that the term $\exp\{-y_n f(x_n)\}$ in (6) is an upper bound of the empirical loss $I(f(x_n) \neq y_n)$. Hence, AdaBoost ensures that the empirical risk will gradually reduce to zero as T approaches infinity.

B. Post-classification – A new perspective to AdaBoost ensemble

Although AdaBoost ensures the reduction of the upper bound (7) of the empirical risk as the number of iterations increases, in practice, AdaBoost must always stop at an allowed maximum number of iterations (say T) so that a composite ensemble classifier can be formed. Following, we will investigate a new way to build an ensemble classifier based on these T classifiers. Given a d-dimensional input vector x, let us consider the following mapping from R^d to R^T:

$$\underline{x} = \Omega(x) = [a_1h_1(x) \;\; a_2h_2(x) \;\; \ldots \;\; a_Th_T(x)]^{\mathrm{t}}, \tag{8}$$

where $\underline{x}$ is referred to as the *weighted hypothesis vector* of the input vector x. The above mapping can be viewed as projecting x to a T-dimensional feature space. We can then cast the original classification problem for S to a new classification problem for $\underline{S} = \{(\underline{x}_1, y_1), (\underline{x}_2, y_2), \ldots, (\underline{x}_N, y_N)\}$ by transforming x_i to its weighted hypothesis vector $\underline{x}_i$. In this paper, we

investigate the problem of finding a good strategy to combine the T weighted hypotheses, a_1h_1, a_2h_2, …, and a_Th_T. We treat this problem as designing a new classifier F: $R^T \rightarrow \{1, -1\}$ for discriminating $\underline{S}$ and we call this problem a *post-classification problem* since F is a classifier of weak classifier outputs, and F is referred to as a post-classifier hereafter.

From the concept of post-classification introduced above, what are inherent interpretations of the original ensemble classifier obtained from AdaBoost as well as each of its learned weak hypotheses? Given a classifier f: $R^d \rightarrow \{1, -1\}$ in the input space, if there is a post-classifier F satisfying that $f = F \circ \Omega$, we call that f can be *equivalently represented* as the post-classifier F and F is the *equivalent* (*post-*) *classifier* of f. Not every classifier in the input space can be equivalently represented as some post-classifier. It is interesting that all the AdaBoost weak hypotheses and AdaBoost ensemble classifier have equivalent post-classifiers:

Property 1: Each of the learned hypotheses, h_j (j = 1, …, T), can be equivalently represented as the post-classifier whose separating boundary is $I^t_j \, \underline{x} = 0$, where I_j is the $T \times 1$ unit vector with its j-th element being one and all other elements being zero.

Property 2: The ensemble classifier of AdaBoost, f_T defined in (5), can be equivalently represented as the post-classifier whose separating boundary is $\boldsymbol{1}^t \, \underline{x} = 0$, where $\boldsymbol{1} = [1\ 1\ \ldots\ 1]^t$ is the $T \times 1$ vector with all its elements being one.

Hence, $\text{sign}(I^t_j \underline{x})$ and $\text{sign}(\boldsymbol{1}^t \underline{x})$ are the equivalent classifiers of h_j (j = 1, …, T) and f_T, respectively, where sign(·) denotes the sign function. An illustration of T = 2 is shown in Figure 1 for explanation. When T = 2, the learned hypotheses h_1 and h_2 can be equivalently represented as the post-classifiers whose boundaries are the Y and X axes, respectively. The output AdaBoost classifier f_T can be represented as the post-classifier whose boundary is the line that passes through the origin, and has a slope of −1.

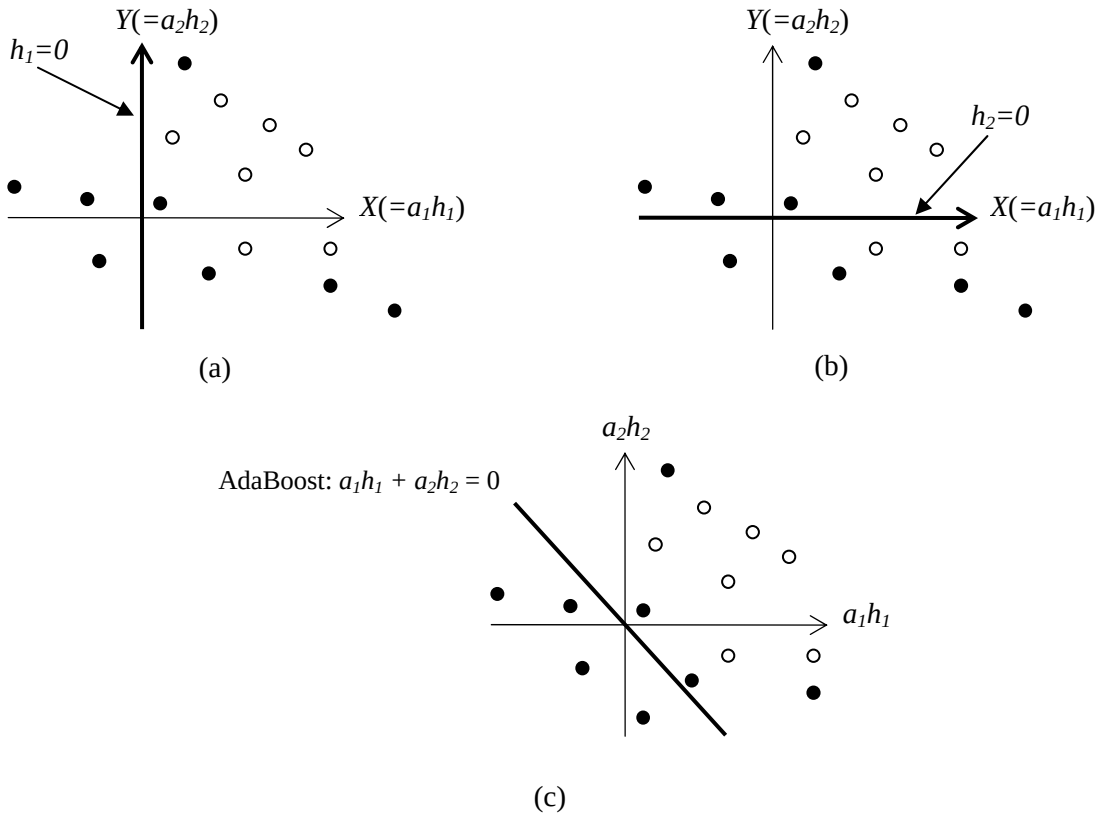

Figure 1. Illustration of the idea of post-classification for $T = 2$. (a) The weak hypothesis h_1 can be equivalently represented as the post-classifier whose classification boundary is the Y-axis. (b) The weak hypothesis h_2 can be represented as the post-classifier whose classification boundary is the X-axis. (c) The ensemble classifier of AdaBoost can be represented as the post-classifier whose boundary is the line that passes through the origin, and has a slope of −1.

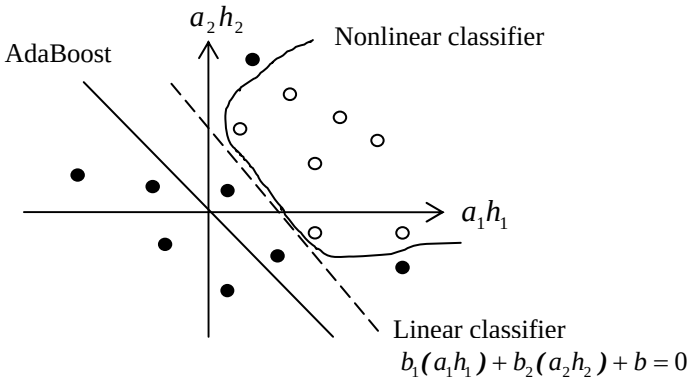

Figure 2. AdaBoost versus nonlinear boost.

Linear classifiers, which have a general form of separating boundaries, $g(\underline{x}) = a^t\underline{x} + b$ ($a \in R^T$ and $b \in R$), have been thoroughly studied in the field of pattern classification [6]. Even when they are not optimal, they form a complete category with effective learning algorithms. The following property clarifies relationships between the linear post-classifiers and the AdaBoost classifiers.

Property 3: Both the equivalent classifiers of h_i (i = 1, …, T) and f_T belong to $\boldsymbol{\Gamma}$, where $\boldsymbol{\Gamma}$ = $\{G: R^T \rightarrow \{1,-1\} \mid G(\underline{x}) = \text{sign}(a^t \underline{x} + b), a \in R^T \text{ and } b \in R\}$ is the set consisting of all the linear classifiers.

Property 3 shows that the learned weak classifiers and the ensemble classifier of AdaBoost are all special cases of the linear classifiers from the post-classification perspective. Note that linear classifiers are themselves special cases of the general classifiers whose boundaries may either be linear or nonlinear. As shown in Figure 2, the classification boundary of a linear classifier may be any line (which has not to pass through the origin) in the plane, and that of a nonlinear classifier is illustrated as a curve.

In this paper, the post-classifiers investigated are not restricted to only be linear, but can be either linear or nonlinear. When a nonlinear classification boundary is employed for post-classification, the boundary can be formulated by nonlinearly projecting $\underline{x}$ into a one-dimensional space by a nonlinear mapping H: $R^T \rightarrow R$, and a general form of a nonlinear discriminant function can be formulated as $F(\underline{x}) = \text{sign}(H(\underline{x}))$. Nonlinear mapping H can be first approximated by a set of basis functions B_j (j = *1, …k*), and then combined to make a decision [6]. By dong so, a nonlinear discriminant function can be approximately formulated as:

$$F(\underline{x}) = \text{sign}(c^t \boldsymbol{B}(\underline{x}) + d), \tag{9}$$

where $c \in R^T$, $d \in R$, and $\boldsymbol{B}(\underline{x}) = [B_1(\underline{x})\ B_2(\underline{x})\ \ldots B_k(\underline{x})]^t$ is a set of nonlinear basis functions. The ensemble of weak hypotheses thus becomes a linear combination of nonlinearly mapped hypotheses vectors. Hence, by employing the post-classification scheme introduced above, nonlinear ensembles of the weak classifiers can be achieved.

Since the output classifier of AdaBoost is a special case from the insight of post-classification, we would like to, as can be expected, seek boundaries that can achieve a better classification performance than the classifier with the original post-classification boundary $\boldsymbol{1}^t\underline{x} = 0$. We will focus on designing post-classifiers for this post-classification problem in the following sections.

Remark: In addition to AdaBoost, many existing boosting methods can be treated as sub-cases under the post-classification scheme. For example, the LP_{REG}-AdaBoost [24], LPBoost [5] and QP_{REG}-AdaBoost [24] methods can be treated as finding post-classifiers whose boundaries are restricted to be linear and pass through the origin. The FloatBoost method [19] removes some insignificant weak hypotheses and linearly combines the remaining hypotheses, it can therefore be viewed as finding a linear post-classifier in which certain linear combination coefficients are zeros.

C. Performance measures review

In order to choose a better classifier to discriminate *S*, one must measure how well a classifier *f* can discriminate *S*, which is typically presented in the form

$$\mathbf{E} = \frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i\right),$$

where *L* is a *loss function* that has been defined in various ways, and **E** is referred to as the

risk function. In our framework, the problem of designing an appropriate classifier f is converted to that of finding the ensemble classifier F for integrating the weighted hypotheses generated by AdaBoost. Therefore, we seek to minimize

$$\mathbf{E} = \frac{1}{N}\sum_{i=1}^{N} L\left(f\left(x_i\right), y_i\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N} L\left(F\left(\Omega\left(x_i\right)\right), y_i\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N} L\left(F\left(\underline{x}_i\right), y_i\right). \tag{10}$$

To design a better ensemble classifier aims to solve the problem of finding a meta-classifier F satisfying $\sum_{i=1}^{N} L\left(F\left(\Omega\left(x_i\right)\right), y_i\right) < \sum_{i=1}^{N} L\left(sign\left(1^t \Omega\left(x_i\right)\right), y_i\right) = \sum_{i=1}^{N} L\left(f_T\left(x_i\right), y_i\right)$, when (10) is chosen as the performance measure.

A commonly used measure is the empirical risk with L being

$$L\left(F\left(\underline{x}_i\right), y_i\right) = I\left(F\left(\underline{x}_i\right) \neq y_i\right). \tag{11}$$

In Neural Networks [11], the loss function is defined as the sum of squared errors,

$$L(F(\underline{x}_i), y_i) = \|F(\underline{x}_i) - y_i\|^2. \tag{12}$$

The above risk functions only focus on training errors, and additional regularization terms can be added to improve generalization abilities. For example, in Neural Networks, energy functions containing complexity penalties $\varepsilon_c(\mathbf{w})$, which depend on the connection weights $\mathbf{w}$ of the network alone, were proposed in [11] to increase the generalization ability:

$$L(F(\underline{x}_i), y_i) = \|F(\underline{x}_i) - y_i\|^2 + \lambda \varepsilon_c(\mathbf{w}), \tag{13}$$

where $\varepsilon_c(\mathbf{w}) = \|\mathbf{w}\|^2$ or $\varepsilon_c(\mathbf{w}) = (w_i/w_0)^2/(1+(w_i/w_0)^2)$, $\mathbf{w}=[w_1\ w_2\ \ldots\ w_{total}]$.

In support vector machine (SVM) [29], an objective function that is defined according to margin maximization is employed to improve generalization ability:

$$E^* = \min_{\underline{\mathbf{w}},b,\boldsymbol{\varepsilon}} \frac{\|\underline{\mathbf{w}}\|^2}{2} + C\sum_{i=1}^{N}\varepsilon_i \tag{14}$$

$$\text{subject to } \begin{matrix} y_i\left(\underline{\mathbf{w}}^t\underline{\mathbf{x}}_i + b\right) \geq 1-\varepsilon_i \forall i \\ \varepsilon_i \geq 0 \forall i \end{matrix}, \tag{14'}$$

where C is a penalty parameter that balances $1/\|\underline{\mathbf{w}}\|$, the margin of the separating hyperplane, and $\sum_{i=1}^{N}\varepsilon_i$, an upper bound of the empirical risk.

D. Finding better post-classifiers

Risk functions introduced above shall be possible choices, but not the only choices, as being performance measures of post-classifiers in our framework. Given a risk function that serves as a common base for classifier comparisons, we try to find properties of the training algorithms that help obtain a better post-classifier than the original AdaBoost ensemble.

In the following, let $\Pi=\{G \mid G:R^T \rightarrow \{1,-1\}\}$ be the functional space consisting of all binary ±1 decision functions. A *hypothesis space* is a subset of Π, and a *machine* Θ is a surjective (i.e., onto) mapping from a parameter space $\boldsymbol{P}$ ($\boldsymbol{P}\subseteq R^m$, $m\in$N) to a hypothesis space Λ ($\Lambda\subseteq\Pi$). Hence, $g = \Theta(\boldsymbol{p})$ is a hypothesis in Λ when some parameter setting $\boldsymbol{p}$ ($\boldsymbol{p}\in\boldsymbol{P}$) is given. A *training algorithm* **M** is referred to as a function that maps any training data set to a parameter setting $\boldsymbol{p}$ ($\boldsymbol{p}\in\boldsymbol{P}$). That is, given a set of training data, the training algorithm **M** finds a parameter $\boldsymbol{p}$ of the machine Θ, which specifies the hypothesis $g = \Theta(\boldsymbol{p})$ in Λ.

Assume that a loss function L in (10) has been chosen to form a performance measure **E** of classification. The following property specifies a simple criterion that is useful for designing post-classifiers.

Property 4: A post-classifier F that is no worse than the output classifier of the AdaBoost f_T can be learnt by a training algorithm **M** if the following conditions hold: (1) A parameter setting $\boldsymbol{p}$ can be given so that $\Theta(\boldsymbol{p}) = \text{sign}(\boldsymbol{1}^t\underline{x})$, and (2) learning algorithm **M** is able to iteratively lowering **E** and finds a local minimum from the initial parameter setting $\boldsymbol{p}$.

Proof: Let $F_{\mathbf{M}}$ be the hypothesis found by **M**. Then $\mathbf{E}(f_T, S, L) = \mathbf{E}(\text{sign}(\boldsymbol{1}^t\underline{x}), S, L) \geq \mathbf{E}(F_{\mathbf{M}}, S, L)$ because $F_{\mathbf{M}}$ is a local minimum of **E** from the initial hypothesis $\text{sign}(\boldsymbol{1}^t\underline{x})$. 7

Although the above property is straightforward, it serves as a general guideline for training a better post-classifier for AdaBoost ensemble. Below, we give an example of the training algorithms that satisfies Property 4.

Property 5: By appropriately setting the initial weights, a multi-layer neural network (NN) can emulate the hypothesis $\text{sign}(\boldsymbol{1}^t\underline{x})$.

Proof: See appendix.

Property 6: A multi-layer NN can serve as a post-classifier that has performance no worse than the AdaBoost ensemble when the performance measure is either the sum of squared errors (12) or is the one with better generalization abilities, (13).

Proof: By using the well-known back-propagation algorithm, a local minimum can be found for the error measure (12). In addition, if the measure with a better generalization ability (13) is used, the weight-decay procedure [12] or the weight elimination procedure [32] can be used to lower the error from an initial setting. 7

E. Post-classification by AdaBoost

Remember that AdaBoost can be treated as a procedure for finding a function *f* to iteratively lowering the empirical risk bound (6). After *T* iterations, this bound has been lowered to $e^* = \sum_{i=1}^{N} \boldsymbol{exp}\left\{ - y_n \sum_{t=1}^{T} a_t h_t(x_i) \right\}$. We will show that, by using AdaBoost as the post-classifier, (6) can be further lowered from e^*.

To distinguish the original AdaBoost, we use *post-AdaBoost* to denote the AdaBoost used for post-classification. According to the post-classification scheme, post-AdaBoost aims to classify the training data $\underline{S} = \{(\underline{x}_1, y_1), (\underline{x}_2, y_2) \ldots (\underline{x}_N, y_N)\}$, where $\underline{x}_i = [a_1h_1(x_i)\ a_2h_2(x_i) \ldots a_Th_T(x_i)]^t$, $i = 1 \ldots N$. To lower (6) by post-AdaBoost, the following two assumptions are made. First, the *T* weak hypothesis $h_1, h_2, \ldots h_T$ satisfy that $\sum_{i=1}^{N} I\left(y_i \neq \sum_{t=1}^{T} a_t h_t(x_i) \right) < N/2$. That is, the output classifier of AdaBoost is better than random guess. This assumption is easy to satisfy since AdaBoost ensures (7) to be lowered to an arbitrarily small positive value when *T* increases, where (7) is an upper bound of the empirical risk, $\frac{1}{N} \sum_{i=1}^{N} I\left(y_i \neq \sum_{t=1}^{T} a_t h_t(x_i) \right)$. Fulfill of this assumption can thus be guaranteed when *T* is large enough. Second, the weak learner used for post-AdaBoost satisfies that the classifier it generates for $\underline{S}$ is $\text{sign}(\boldsymbol{1}^t \underline{x})$. This assumption is also easy to satisfy by forcing the weak learner used in post-AdaBoost outputs $\text{sign}(\boldsymbol{1}^t \underline{x})$ in the first iteration.

By the above assumptions, the initial value of the error measure (6) achieved by post-AdaBoost is $\sum_{i=1}^{N} \boldsymbol{exp}\left\{ - y_i \, sign\left(1^t \underline{x}_i \right) \right\}$, which is equal to e^* defined above. In addition, by the first assumption, enforcing the weak hypothesis generated in the first iteration of

post-AdaBoost to be sign($\mathbf{1}^t \underline{x}$) ensures ε_1 to be smaller than 0.5 because $\varepsilon_1 = \sum_{n=1}^{N} \frac{1}{N} I\left(y_n \neq \underline{h}_t(\underline{x}_n)\right) = \sum_{n=1}^{N} \frac{1}{N} I\left(y_n \neq sign\left(1^t \underline{x}_n\right)\right)$. When the weak learner used for AdaBoost remains to perform better than random guess for the rest iterations, the error measure (6) can then be lowered from the initial value e^* by post-AdaBoost because the AdaBoost procedure essentially continues to find f to lower (6). Since (6) is an upper bound of the empirical risk, similar to AdaBoost, post-AdaBoost can also reduce the empirical risk to zero as the number of iterations approaches infinity.

Remark: Note that AdaBoost itself can continue to reduce (6). It is thus worth to compare AdaBoost and post-AdaBoost on how (6) is reduced. Instead of (7), the empirical risk bound in post-AdaBoost is $\sum_n \exp\left\{-y_n f\left(x_n\right)\right\} = \sum_n \exp\left\{-y_n \sum_t \underline{a}_t \underline{h}_t(\underline{x}_n)\right\}$, where $\underline{a}$ and $\underline{h}$ denote the combination coefficients and weak hypotheses generated by post-AdaBoost, respectively. Assume that AdaBoost continues to run further T' iterations (in addition to the original T iterations), then (6) is reduced to $\sum_{i=1}^{N} \boldsymbol{exp}\left\{-y_n \sum_{t=1}^{T+T'} a_t h_t\left(x_i\right)\right\}$. In contrast, when T' iterations have been run for post-AdaBoost, the value of (6) becomes $\sum_{i=1}^{N} \boldsymbol{exp}\left\{-y_n \sum_{t=1}^{T'} \underline{a}_t \underline{h}_t\left(\underline{x}_i\right)\right\}$, where $\underline{x}_i = [a_1 h_1(x_i)\ a_2 h_2(x_i)\ \ldots\ a_T h_T(x_i)]^t$. From the above analysis, post-AdaBoost provides an alternative to minimize (6) by linearly combine a new set of hypotheses, $\underline{h}_1, \underline{h}_2 \ldots \underline{h}_{T'}$, instead of working on the original weak hypotheses, $h_{T+1}, h_{T+2} \ldots h_{T+T'}$, generated by AdaBoost. However, it is difficult to theoretically justify which one (AdaBoost or post-AdaBoost) is better. The appropriateness of choosing either one of them shall be data dependent.

III. SVM-based Post Classifier

We have shown that a training algorithm, which is able to improve the classifier performance from an initial setting, is likely to generate a better post-classifier if sign($\boldsymbol{1}^t\underline{x}$) can be explicitly encoded as the initial setting. We have shown that both multi-layer NN and post-AdaBoost can refine the original AdaBoost ensemble when the performance measures are (12)/(13) and (6), respectively. However, these algorithms simply improve the performance from the initial settings, but may be trapped in local minimums. As well, they cannot ensure that the results are no worse than that of each weak hypothesis. To avoid these difficulties, let us investigate training algorithms that find not only a local minimum but the global minimum.

A. Finding the best post-classifiers

When a training algorithm is capable of finding the global minimum, a significant distinction is that we do not need to specify a explicit parameter setting $\boldsymbol{p}$ so that the hypothesis sign($\boldsymbol{1}^t\underline{x}$) can be emulated as depicted in Property 4. We only need to ensure that the hypothesis space contains sign($\boldsymbol{1}^t\underline{x}$), since the resulting classifier is independent of the initial parameters. With such strong training algorithms, the following property holds.

Property 7: The optimal post-classifier F whose performance is no worse than AdaBoost ensemble can be learnt by a training algorithm **M** if the following conditions are satisfied: (1) The hypothesis space Λ contains sign($\boldsymbol{1}^t\underline{x}$), and (2) **M** is able to find the global minimum for the performance measure **E**.

The above property serves as the main criterion for choosing and designing an appropriate post-classifier in our work. In addition, the following property gives further motivation for selecting a training algorithm that is not trapped in local minimums.

Property 8: [*Continuing from Property 7*] If **M** further satisfies that its hypothesis space Λ contains $\text{sign}(I^{t}_{j}\ \underline{x})$ for all $i = 1, \ldots, T$, then **M** can find a post-classifier whose performance is no worse than those of all the classifiers generated in the AdaBoost process, including the AdaBoost ensemble f_T and all weak hypotheses h_i ($i = 1, \ldots, T$).

Property 8, an extension of Property 7, shows that a training algorithm capable of finding the global minimum is likely to produce a post-classifier not only better than the AdaBoost ensemble, but also better than all weak hypotheses. Note that this does not hold for a training algorithm that finds local minimums because it merely ensures obtaining a better a classifier than the initial one. It is difficult to find a post-classifier that is better than all classifiers generated during the AdaBoost process since only one of the classifiers is likely to be encoded as the initial setting for starting the training algorithm. [1]

B. Post-classifying using linear SVM

By combining Property 8 and Property 3, we obtain the following property in association with linear classifiers.

Corollary 1: If **M** can find the global minimum and $\Gamma \subseteq \Lambda$, (i.e., the set consisting of all linear classifiers is a subset of the hypothesis space Λ), then **M** can find a post-classifier no worse than the AdaBoost ensemble and all weak hypotheses.

Corollary 1 shows that, if a training algorithm can find the best linear classifier, then it can refine AdaBoost ensemble. Since linear classifiers have been extensively investigated and many associated effective learning algorithms have been proposed, this corollary provides very useful guidelines for choosing post-classifiers from existing approaches. In a special

[1] A possible way to overcome this difficulty is to re-start the training algorithm multiple times and find the smallest local minimum. However, this incurs a large time complexity.

circumstance in which the post-classification problem encountered is linearly separable, standard perceptron can ensure finding the global minimum in finite steps if the empirical risk (11) is chosen as the performance measure. In general circumstances in which the post-classification problem is not linearly separable, the pocket perceptron [23] that globally minimizes the empirical risk (11) can be used. Nevertheless, minimizing the empirical risk usually causes an over-fitting effect. We will investigate training algorithms that are capable of finding the global minimum for a performance measure that has better generalization ability.

Among the existing approaches, linear SVM [4] is a training algorithm that finds the best linear classifier by minimizing risk function (14) which has a better generalization ability than the empirical risk. In SVM [4], optimization problem (14) (subject to the constraint (14')) can be transformed into a dual form for which the global minimum can be obtained by solving a convex quadratic programming problem. After (14) is solved, $sign\left(\underline{\boldsymbol{w}}^t \underline{\boldsymbol{x}}_i + b\right)$ serves as the discriminant function of linear SVM.

Note that the QP_{REG}-AdaBoost method presented in [24] minimizes exactly the same quadratic programming optimization problem as linear SVM by further imposing the constraint b=0 in (14'). Hence, QP_{REG}-AdaBoost can be treated as a special case of linear SVM in the post-classification scheme by restricting the classification boundary of the post-classifier to pass through the origin. [2]

[2] Note that the same arguments hold for the LP_{REG}-AdaBoost method [24] and another type of SVM – the linear-programming SVM ([4], Section 6.1.3). LP_{REG}-AdaBoost method can be treated as a special case of linear-programming SVM in the post-classification scheme by restricting the classification boundary of the post-classifier to pass through the origin.

C. Linear versus nonlinear

When linear SVM is used for post-classification, the hypothesis space considered is Γ. It is worth noting that Corollary 1 holds not only when the hypothesis space is Γ, but also for a larger hypothesis space containing Γ. We can expect that the performance may be further enhanced if a larger hypothesis space is explored, as clarified by the following property.

Property 9: Consider two training algorithms, **M** and **M'**, that find the global minimum for the same performance measure. If $\Lambda \subseteq \Lambda'$, i.e., the hypothesis space of **M** is contained in the hypothesis space of **M'**, **M'** can find a post-classifier no worse than that found by **M**.

Note that the hypothesis space of linear SVM can be significantly enlarged by projecting $\underline{x}$ into a high-dimensional space via an implicit mapping of which the inner product can be effectively computed by using Mercer's kernel, and a nonlinear SVM is accordingly formed [4]. Below, we will first review nonlinear SVM, and then investigate how nonlinear SVM can be employed for post-classification.

D. Review of nonlinear SVM

Let the mapping from the weighted hypothesis space to the high-dimensional space be Φ, then the optimization problem of nonlinear SVM is

$$E^{**} = \min_{\underline{\mathbf{w}},b,\boldsymbol{\varepsilon}} \frac{\|\underline{\mathbf{w}}\|^2}{2} + C\sum_{i=1}^{N}\varepsilon_i = \min_{\underline{\mathbf{w}},b,\boldsymbol{\varepsilon}} \xi(\underline{\mathbf{w}}, C, \varepsilon) \tag{15}$$

$$\text{Subject to } \begin{matrix} y_i\left(\underline{\boldsymbol{w}}^t\Phi(\underline{\boldsymbol{x}}_i)+b\right) \geq 1-\varepsilon_i \forall i \\ \varepsilon_i \geq 0 \forall i \end{matrix}. \tag{15'}$$

The solution to the above optimization problem is

$$\boldsymbol{w}^* = \sum_{i=1}^{N} y_i \alpha_i^* \Phi(\underline{\boldsymbol{x}}_i), \tag{16}$$

and the optimal classifier is

$$\text{sign}(\left(\boldsymbol{w}^*\right)^t \underline{\boldsymbol{x}} + b) \;=\; \text{sign}(\sum_{i=1}^{N} y_i \alpha_i^* K(\underline{\boldsymbol{x}}_i, \underline{\boldsymbol{x}}) + b). \tag{17}$$

Performing a linear classification in high-dimensional space is equivalent to performing a nonlinear classification in the original space, where the nonlinear classifiers are constructed by linearly combining evaluations of the selected kernel (or basis) function, as shown in (17). Commonly used kernel functions include polynomial kernels $K^{\boldsymbol{P},h}$ and radial basis kernels $K^{\boldsymbol{R},\gamma}$:

$$K^{\boldsymbol{P},h}(z_1, z_2) = (z_1{}^t z_2 + 1)^h, \tag{18}$$

$$K^{\boldsymbol{R},\gamma}(z_1, z_2) = \exp(-\gamma \|z_1 - z_2\|^2). \tag{19}$$

E. Post-classification by nonlinear SVM

Nonlinear SVM is a powerful classification tool and has worked well for many applications. Nevertheless, when such a nonlinear classification framework is used, it can not guarantee that sign($\boldsymbol{1}^t \underline{x}$) is contained in the hypothesis space, which makes it troublesome for effective post-classification.

Fortunately, a recent study [16] shows that linear SVM can be emulated by nonlinear SVM when certain asymptotic behavior is considered, if the radial basis kernel (19) is employed. The limiting SVM radial-basis kernel classifier as $\gamma \to 0$ can be the same as the SVM linear kernel classifier in some condition. We can adopt nonlinear SVM for post-classification that satisfies Corollary 1 if we consider Property 10.

Property 10 (Keerthi and Lin, 2003 [16]) Assume that radial basis kernel (19) has been used to construct a nonlinear SVM. If $\gamma \to 0$ and $C = C'\gamma$ (where C' is fixed) then the nonlinear SVM classifier converges to the linear SVM classifier with penalty parameter C'.

Property 10 shows that, in effect, the performance measure E^{**} defined in (15) degenerates to E^{*} in (14) when the asymptotic behaviors $\gamma \to 0$ and $C = C'\gamma$ are tracked. Hence, if we apply model selection to nonlinear SVM in which the model parameters γ and C are selected to minimize E^{**} in (15) for solving the optimization problem

$$\min_{\gamma,C} E^{**} = \min_{\gamma,C} \min_{\underline{\mathbf{w}},b,\boldsymbol{\varepsilon}} \xi\left(\underline{\mathbf{w}},C,\varepsilon\right) \tag{20}$$

subject to the constraints in (15'), then nonlinear SVM subsumes linear SVM. Specifically, linear SVM combined with model selection is equivalent to solving the minimization problem $\min_{\gamma\to 0, C=C'\gamma} E^{**}$. The hypothesis space of linear SVM can therefore be thought of as being contained in the hypothesis space of nonlinear SVM which solves a more general optimization problem $\min_{\gamma,C} E^{**}$. According to Property 9, nonlinear SVM may be able to find a better minimum for the performance measure ξ.

IV. Implementation Issues and Experiment Results

Although in theory, there is no need to consider linear SVM if complete model selection using the radial-basis kernel has been conducted, a dense sampling may be involved in model selection for approximating the asymptotic behavior of linear SVM. In practice, it is not necessary to restrict model selection to only nonlinear SVM since performing an additional model selection for linear SVM lays only a small overhead. Hence, a practical and useful alternative is to use both linear and nonlinear SVMs for model selection. By doing so, ξ in

(20) serves as a common performance measure.

A. Main algorithm

From the above, we can find the best post-classifier of the optimization problem defined in (20) by using SVM. This classifier ensures that ξ is minimized in a hypothesis space containing all the classifiers generated by AdaBoost. By treating ξ as a common performance measure, we then guarantee that the method which uses SVM for post-classification performs no worse than AdaBoost.

However, ξ may not reflect the real classifier performance directly. In practice, the cross-validation (CV) error was mainly used to measure classifier performances, and the model parameters of SVM (γ and C) were selected based on the CV error, instead of ξ. Our algorithm is slightly modified accordingly such that ξ is minimized for training each individual SVM but the CV error is minimized for model selection, as done in most SVM approaches. The reason is because (i) the CV-error directly relates to the generalization ability of a classifier, and (ii) well-established SVM libraries usually provide model selection combined with cross validation for SVM training, which makes our algorithm easy to implement. [3] Our main algorithm that uses SVM as a post-classifier is shown below.

Algorithm – Nonlinear Boost using SVM

0. **Input**: S={($\boldsymbol{x}_1$, y_1), ($\boldsymbol{x}_2$, y_2), … ,($\boldsymbol{x}_N$, y_N)}, number of iterations T, a set of linear SVM model parameters $\boldsymbol{\tau}_l$ = {C_1, C_2, … } and a set of radial-basis nonlinear SVM parameters $\boldsymbol{\tau}_{nl}$ = {(C_1, Υ_1), (C_2, Υ_2) …}. Let $\boldsymbol{\tau} = \boldsymbol{\tau}_l \cup \boldsymbol{\tau}_{nl}$. **Initializ**e: e^*=0.
1. Use AdaBoost to find h_1, h_2,…, h_T and coefficients a_1, a_2,…, a_T. Let $\boldsymbol{\Omega}(\boldsymbol{x})$ = [$a_1h_1(\boldsymbol{x})$

[3] In this work, the C/C++ SVM library LibSVM [35] was used.

$a_2h_2(\boldsymbol{x}) \ldots a_Th_T(\boldsymbol{x})]^t$ and $\underline{S}=\{(\boldsymbol{\Omega}(\boldsymbol{x}_1), y_1), (\boldsymbol{\Omega}(\boldsymbol{x}_2), y_2), \ldots, (\boldsymbol{\Omega}(\boldsymbol{x}_N), y_N)\}$.

2. **For** a list of model parameters $\tau_i \in \boldsymbol{\tau}$, i=1,2,…

   3.1. Use the SVM with model parameter τ_i to train $\underline{S}$ by solving (14) when τ_i represents a model parameter of linear SVM, or (15) otherwise. Compute the cross validation (CV) error e.

   3.2. **If** $e < e^*$ **then** set $e^* = e$ and $\tau^* = \tau_i$.

3. Set SVM's model parameters to τ^* and output the trained classifier.

In the above algorithm, AdaBoost and SVM are systematically combined through post-classification to obtain an integrated classifier. There are many ways to define the CV error, and the one used in our experiments will be described in Section IV.B. Although our algorithm cannot ensure that the CV error minimized is no larger than that of AdaBoost because the CV error is not consistently employed as a common performance measure for both individual SVM construction and model selection, employing CV error for model selection is a popular strategy and is effective in creating classifiers with high generalization abilities [6][24][1]. Experiment results show that this algorithm can successfully improve the generalization performance of AdaBoost for un-trained test data in cross validation.

Remark: Our main algorithm uses different error measures, ξ and CV error, for individual SVM training and model selection, respectively. If ξ and CV error are 'consistent' measures – consistent in the sense that one is an increasing mapping of the other, they can be treated as a common measure and our method ensures finding the optimal classifier for either measures. Although in theory, it is difficult to relate general CV error with the SVM error measure ξ defined in (15), there is much work dealing with relationships between ξ and the leave-one-out (*loo*) error [3][8][18] that is a special type of the CV error in case that the number of folds equals to that of the training examples. Some bounds of *loo* error can be

derived and represented in terms of SVM parameters, which help model selection for SVM. For example, an informative *loo*-error bound for the radial-basis-kernel SVM is ([3], eq. 3.3)

$$loo \leq 2(R^2+\Delta/C)\xi, \tag{21}$$

where Δ is a positive constant close to one and R is the radius of the smallest sphere containing all $\Phi(\underline{x}_i)$ in the high-dimensional feature space. Solving ξ for the optimization problem (20) thus restricts *loo* error to be bounded by a factor $2(R^2+\Delta/C)$. Although *loo* error and ξ cannot be shown to be exactly consistent, they are somewhat correlated. In addition, note that Property 10 also holds when the error measure is *loo* error [16]. The above analysis gives a more concrete reason why the main algorithm is effective for finding a post-classifier better than AdaBoost ensemble.

B. Experiment results

We performed some experiments to compare the performances of AdaBoost and our method. Seven benchmark datasets, bupa, diabetes, ionosphere, sonar, splice, titanic and wdbc, from UCI machine learning repository [33] and Delve assessment datasets [34] were used in these experiments. For small datasets (referred to as the datasets whose number of training examples is smaller than 600), bupa, ionosphere, sonar and wdbc, we perform 5-fold cross-validations. For the remaining large datasets, diabetes, splice and titanic, we perform 10-fold cross-validations.

Cross-validation accuracy was used to select the best model parameters, which is defined as follows. Typically, in n-fold cross validation, n examinations are performed and each of them involves $(n-1)/n$ data for training and the remaining $1/n$ data for test. The cross-validation accuracy is defined as the ratio of the number of correctly classified test data

to the number of test data (in %) in an examination (i.e., in a fold). The mean (or average) of the cross-validation accuracies of the *n* examinations is used for model selection.

The algorithm presented in Section IV.A was used for training the classifiers. Since the algorithm (nonlinear boost using SVM) is designed by integrating AdaBoost and SVM, we use AdaBoost+SVM to denote it. We have also to choose the weak learners used in the algorithm. In our experiments, Radial Basis Function networks (RBF nets) [11] and SVM [4] were used as weak learners. Because SVM also serves as post-classifier, to distinguish with them, we use WeakLearner+AdaBoost+SVM to denote the algorithms using different weak learners. Hence, RBF+AdaBoost+SVM and SVM+AdaBoost+SVM denote the nonlinear boost training algorithms whose weak learners are RBF nets and SVM, respectively. In both cases, we have empirically found that the performance is better when the weak learner outputs a real value, instead of a binary value. Hence, we use RBF-net regression [24] and SVM regressor [20] as weak learners in our experiments. Nevertheless, SVM used for post-classification remains to output binary values.

In the first experiment, RBF nets were used as weak hypotheses for AdaBoost. The algorithm RBF+AdaBoost+SVM was implemented to train the seven datasets. The iteration number T was set to 200, and thus 200 weak RBF classifiers were generated. For selecting the best RBF model we do the model selection on both the number of centers n_c and the number of iteration steps n_L for adapting the RBF centers. Since only two integer parameters have to be tuned in the model-selection process in this case, we perform full search for finding the best pair of parameters, where both n_c and n_L are selected from {2, 4, 6, 8, 10}. For each pair of n_c and n_L, model selection for the post-classifier SVM were performed by sampling the regularization constant C in (15) and the kernel parameter γ in (19) from $\{2^{-7}, 2^{-6}, \ldots, 2^{7}\}$. To compare with the original AdaBoost algorithm, AdaBoost using

RBF nets as its weak learner (denoted by RBF+AdaBoost) was also implemented. Model selection on n_c and n_L was performed for RBF+AdaBoost to train the seven datasets too.

Table 1 shows the performances of our method and AdaBoost, where the mean and standard deviations (in %) of the cross validation accuracies are listed. It can be seen from Table 1 that RBF+AdaBoost+SVM outperforms RBF+AdaBoost on most (six of the seven) datasets, as its means are larger and variances are smaller. This experiment shows that nonlinear boost has a great possibility to improve the performance of AdaBoost.

In the second experiment, we used SVM to investigate the effect of strong hypotheses for nonlinear boosting. It is known that, if a 'strong' classifier is used as 'weak' learner for AdaBoost, training data may be over-fitted and the ensemble classifier may have a limited performance improvement than the weak classifiers. Nevertheless, when strong hypotheses are combined nonlinearly, the ensemble hypothesis is likely to be improved more significantly. We compare three SVM-based classification methods: single SVM classifier, AdaBoost using SVM as weak learner (referred to as SVM+AdaBoost), and the nonlinear boost method SVM+AdaBoost+SVM.

In both SVM+AdaBoost and SVM+AdaBoost+SVM, we combined T=50 weak hypotheses. For the model selection of SVM+AdaBoost+SVM, two SVM parameters C and γ in association with both weak learner and post-classifier have to be investigated. Accordingly, the model selection is done through a grid-search in a 4-D parameter space. To avoid the time complexity, we used a coarse-to-fine strategy combined with branch-and-bound for model parameter search (which is similar to the method introduced in [17]), so that we need not examine all grid points.

The experiment results are shown in Table 2, which reveals that SVM+AdaBoost performs slightly better than single SVM classifier. The nonlinear boost method

SVM+AdaBoost+SVM further improves the performance of SVM+AdaBoost and achieves the best cross validation accuracy (i.e., the largest mean and the smallest standard deviation) among the three methods for all the datasets.

On the datasets bupa and sonar, the hypothesis of the first round obtained by SVM+AdaBoost can classify all training data correctly, and thus the algorithm of SVM+AdaBoost stops immediately after the first round. Consequently, on these two datasets, SVM+AdaBoost and single SVM classifier have identical performances, and the post-classifier of SVM+AdaBoost+SVM only needs to solve a 1-D classification problem. In this case, it is interesting that nonlinear boost can still improve the performance of AdaBoost.

An interesting issue worth of being addressed is that, is nonlinear classifier really helpful in post-classification? To answer this question, remember that we have performed model selections for both linear SVM and nonlinear SVM in our main algorithm. In all the above experiment results, the best parameters come from nonlinear SVM but not linear SVM (sometimes they tie), which shows that a training algorithm whose hypothesis space contains nonlinear classifiers can indeed improve the ensemble performance for the training algorithm whose hypothesis space contains only linear classifiers.

V. Further Remarks

[**Improving Other Boosting Methods**] A characteristic of the post-classification scheme proposed in this paper is that, except AdaBoost, it also has potential to improve other boosting methods. Many modified versions of AdaBoost (e.g., $\text{AdaBoost}_{\text{reg}}$ [24], RealBoost [19], FloatBoost [19]) generate weak hypotheses different from those generated by AdaBoost. These hypotheses are linearly combined in these methods and thus post-classification can be used for conducting a possibly better combination of these weak hypotheses. Following the

same arguments in Sections 2 and 3, Property 1 to Property 10 and Corollary 1 hold too. A post-classifier can therefore be created to construct a composite ensemble classifier that is no worse than the original ensemble in which the weak hypotheses are combined linearly.

[**Time Complexity for Classification**] Compared to AdaBoost, nonlinear boost needs to perform a further post-classification when it is used for pattern classification. Nevertheless, the overhead is relatively insignificant because AdaBoost itself needs to perform classifications for multiple times because multiple weak classifiers are generated. Let a single classification step be counted as a major operation. In AdaBoost, T classifications have to be performed, while T+1 classifications have to be performed for nonlinear boost. Hence, nonlinear boost adds a very limited overhead to AdaBoost in major operations.

On the other hand, for the same iteration times, nonlinear boost usually performs better than AdaBoost. An illustrative example can be found in Figure 4. Hence, $\underline{T}$ iterations may be enough for nonlinear boost to achieve the same performance of AdaBoost, where $\underline{T} < T$. In this situation, nonlinear boost may even be faster in classification than AdaBoost.

[**Multi-Class Problem**] The post-classification scheme can be straightforwardly generalized for multi-class problems by using a multi-class classifier as the post-classifier or using one-against-one strategy to combine multiple two-class classifiers as a multi-class classifier. However, in multi-class AdaBoost, the output label is decided by combining several discriminate functions linearly, and then selecting the highest-score one, which yields a piecewise-linear classification boundary. Unlike the two-class case, existing multi-class training methods cannot ensure finding the global minimum of an error measure in a hypothesis space that contains all the piecewise-linear classifiers (to our best knowledge). In contrast, in the two-class case, the functional space of piecewise-linear classifiers degenerates to that of linear classifiers, and SVM is an example that finds the global

minimum for an error measure with good generalization abilities in a hypothesis space consisting of all linear classifiers.

On the other hand, if we loose the requirement to find local minimums, then multi-layer NN ensures to improve the original piecewise-linear classifier based on the error measures (12) or (13), because a piecewise-linear classifier can be initially encoded to emulate NN by setting appropriate weights.

VI. Conclusions and Future Work

In this paper, we propose a new classifier ensemble scheme – post-classification, which provides a general perspective on combining weak hypotheses generated by boosting. We have derived useful properties on how to combine weak hypotheses for creating a powerful composite ensemble that performs better than the original AdaBoost ensemble. By using SVM as post-classifier, we have also shown that AdaBoost and SVM can be effectively combined in the proposed post-classification scheme to achieve an effective training algorithm for classifier synthesis. Experiment results show that our method can considerably improve AdaBoost's performances.

Some future works are addressed below.

Sometimes we hope to reduce the number of weak hypotheses to be combined, so that a more efficient classifier can be synthesized. In the post-classification scheme, this can be treated as designing classifiers whose boundaries are orthogonal to the subspace formed by the weak hypotheses that are preserved. Previous work such as FloatBoost [19] provides solution to this problem, but the preserved classifiers are combined following the same linear combination method of AdaBoost. By combining some feature-selection method [13], we plan to study this problem in a more general manner based on the post-classification scheme.

Nonlinear boost also has potential to be used for one-class classification problems. In particular, nonlinear post-classification has potential to be applied for improving boosting-like one-class classifiers, such as the one-class-leveraging algorithm [25].

In addition, the post-classification scheme can be applied not restricted to the classifiers generated by boosting processes. The framework of this paper can be used for multiple classifier systems [7] and is helpful in understanding how to design nonlinear combinations better than linear combinations.

Appendix [Proof of Property 5]

There are many ways to construct a multi-layer Neural Network to emulate the hypothesis sign($\boldsymbol{1}^t\underline{x}$). The simplest way is to construct a two-layer network, where the input layer contains N nodes and the output layer contains a single node, as shown in Figure 3(a). The activation function of the output node is a sigmoid one, $\boldsymbol{o}(x) = 1/(1+e^{-ax})$, with a set very high so that the sign function can be simulated. In this network, if we set all the connection weights to one, then the network represents the hypothesis sign($\boldsymbol{1}^t\underline{x}$). This simple network can be extended to a three-layer one, as shown in Figure 3(b), where the hidden layer contains l nodes and the output layer still contains a single node. The lT weights from the input nodes to the hidden nodes are all set to one, and the l weights from the hidden nodes to the output node are all set to $1/l$. To verify that this three-layer network emulates sign($\boldsymbol{1}^t\underline{x}$) note that the output is sign($\sum_{i=1}^{l}(1/l)$ sign($\boldsymbol{1}^t\underline{x}$)) = sign($\boldsymbol{1}^t\underline{x}$). 7

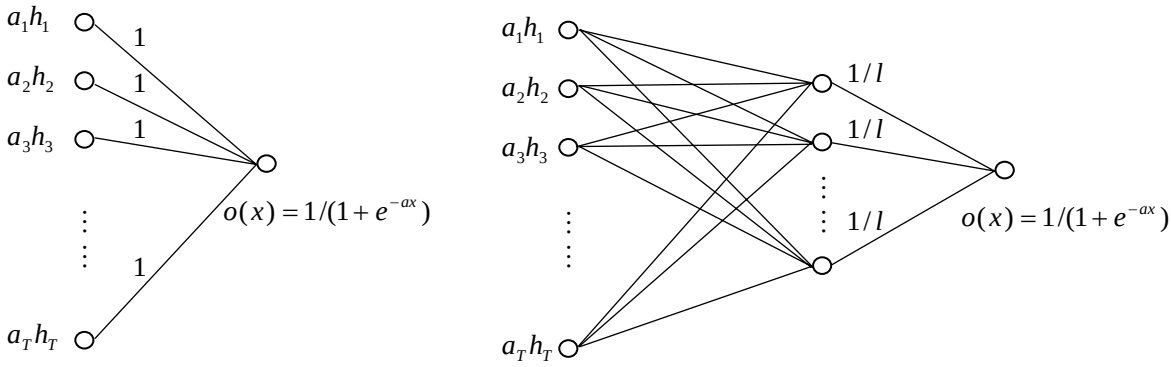

Figure 3. Emulating *sign(1^{t}x)* by multi-layer Neural Networks, in which *a* is set very high so that the sign function can be simulated. (a) A two-layer example. (b) A three-layer example in which the *lT* weights from the input nodes to the hidden nodes are all set to one.

References

[1] J.-H. Chen and C.-S. Chen, "Fuzzy Kernel Perceptron," *IEEE Transactions on Neural Networks*, Vol. 13(6), pp. 1364-1373, 2002.

[2] T. Choudhury, et al., "Boosting and Structure Learning in Dynamic Bayesian Networks for Audio-Visual Speaker Detection," *Proceedings of International Conference on Pattern Recognition*, Quebec City, Canada, pp. III:789-794, 2002.

[3] K. M. Chung, W. C. Kao, and C. L. Sun et. al, "Radius Margin Bounds for Support Vector Machine with the RBF Kernel," *Neural Computation*, 15, pp. 2643-2681. 2003.

[4] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, UK, 2000.

[5] A. Demiriz, K. Bennett and J. Shawe-Taylor, "Linear Programming Boosting via Column Generation," *Machine Learning*, vol. 46, pp. 225-254, 2002.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, Second Edition, John Wiley & Sons, Inc., 2001.

[7] R. P. W. Duin, "The Combining Classifier: To Train or Not to Train," *Proceedings of ICPR'2002*, Quebec, Canada, 2002.

[8] A. Elisseeff and M. Pontil, "Leave-one-out Error and Stability of Learning Algorithms with Applications," in J. Suykens, G. Horvath and S. Basu et. Al eds, *Learning Theory and Practice*, NATO ASI Series, IOS Press, Amsterdam: Washington, DC, 2002.

[9] Y. Freund and R. E. Schapire, "A Decision-theoretic Generalization of On-line Learning and An Application to Boosting," *Journal of Computer and System Sciences*, 55(1), pp. 119-139, 1997.

[10] G. D. Guo and H. J. Zhang. "Boosting for Fast Face Recognition," *Proceedings of IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems* (RATFG-RTS 2001), 2001.

[11] S. Haykin, *Neural Networks, A Comprehensive Fundation*, Prentice Hall, NJ, 1994.

[12] G. E. Hinton and S. J. Nowlan, "How Learning Can Guide Evolution," *Complex Systems*, Vol. 1, pp. 495-502, 1987.

[13] F. J. Iannarilli Jr. and P. A. Rubin, "Feature Selection for Multiclass Discrimination via Mixed-integer Linear Programming," *IEEE Transactions on Pattern Analysis and Machine Intelligences*, Vol. 25, No. 6, pp. 779-783, 2003.

[14] R.D. Iyer, D.D. Lewis and R.E. Schapire et al., "Boosting for Document Routing," *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 2000.

[15] M. Kearns and L. Valiant, "Cryptographic Limitations on Learning Boolean Formulae and Finite Automata," *Journal of the ACM*, vol. 41, pp. 67-95, 1994.

[16] S. S. Keerthi and C. J. Lin, "Asymptotic Behavior of Support Vector Machine with Gaussian Kernel," *Neural Computation*, Vol. 15, pp. 1667-1689, 2003.

[17] J.-H. Lee and C.-J. Lin, "Automatic Model Selection for Support Vector Machines," *http://www.csie.ntu.edu.tw/cjlin/papers/modelselect.ps.gz*, 2000.

[18] M. M. S. Lee, S. S. Keerthi, and C. J. Ong, et al., "An Efficient Method for Computing

Leave-One-Out Error in Support Vector Machine," to appear in *IEEE Transactions on Neural Networks*.

[19] S.Z. Li, Z.Q. Zhang and Harry Shum et al., "FloatBoost Learning for Classification," *Proceedings of the 16-th Annual Conference on Neural Information Processing Systems NIPS 2002*, Vancouver, Canada, 2002.

[20] S.-P. Liao, H.-T. Lin, and C.-J. Lin, "A note on the decomposition methods for support vector regression," *Neural Computation*, 14(2002), 1267-1281

[21] R. Lockton and A. W. Fitzgibbon, "Real-time Gesture Recognition Using Deterministic Boosting," *Proceedings of the British Machine Vision Conference*, 2002.

[22] R. Meir and G. Rätsch, "An Introduction to Boosting and Leveraging," in S. Mendelson and A. Smola, eds., *Advanced Lectures on Machine Learning*, LNCS, pages 119-184. Springer, 2003.

[23] M. Muselli, "On Convergence Properties of Pocket Algorithm," *IEEE Transactions on Neural Networks*, Vol. 8(3), pp. 623-629, 1997.

[24] G. Rätsch, T. Onoda and K. –R. Müller, "Soft Margins for AdaBoost," *Machine Learning* , Vol. 42, pp. 287-320, 2001.

[25] G. Rätsch, S. Mika, B. Schölkopf and K. –R. Müller, "Constructing Boosting Algorithms for SVMs: An Application to One-Class Classification," *IEEE Transactions on Pattern Analysis and Pattern Analysis*, Vol. 24, pp. 1184-1197, 2002.

[26] R. E. Schapire and Y. Singer, "BoosTexter: A Boosting-based System for Text Categorization," *Machine Learning*, Vol. 39(2/3), pp. 135-168, 2000.

[27] R. Schapire, "The Boosting Approach to Machine Learning, An Overview," *Proceedings of MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[28] R. A. Servedio, "Smooth Boosting and Learning with Malicious Noise," *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*, 1999, pp. 278-285.

[29] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[30] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[31] H. Schwenk, "Using Boosting to Improve a Hybrid HMM/Neural Network Speech Recognizer," *Proceedings of IEEE International Conference On Acoustics, Speech, and Signal Processing* (ICASSP 99), Vol. 2, pp. 1009-1012, 1999.

[32] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by Weight-elimination with Application to Forecasting," *Advances in Neural Information Processing Systems 3* (R. Lippmann et al. eds), Morgan Kaufmann, CA, pp. 875-882, 1991.

[33] http://www.ics.uci.edu/~mlearn/MLRepository.html

[34] http://www.cs.toronto.edu/~delve/data/datasets.html

[35] http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html

Table 1. Comparisons between AdaBoost and Nonlinear Boost. We perform cross-validations on 7 data sets. The means and the standard deviations of the test accuracy are shown in percent (%).

Cross-validation accuracy (%)

	AdaBoost (RBF+AdaBoost)		Nonlinear Boost (RBF+AdaBoost+SVM)	
bupa	71.01	5.423	74.20	4.037
diabetis	74.74	4.596	77.11	1.785
ionosphere	84.86	4.585	86.29	4.199
sonar	91.71	3.699	91.22	4.253
splice	91.67	1.254	92.43	1.129
titanic	79.09	4.166	79.18	3.742
wdbc	95.22	1.481	96.11	1.200

Table 2. Performance comparisons among three SVM-based classification methods: single SVM classifier, SVM+AdaBoost, and SVM+AdaBoost+SVM.

Cross-validation accuracy (%)

	SVM		AdaBoost (SVM+AdaBoost)		Nonlinear Boost (SVM+AdaBoost+SVM)	
bupa	63.77	3.55	63.77	3.55	64.93	3.14
diabetis	77.76	3.80	77.89	3.91	78.68	3.15
ionosphere	92.88	3.19	94.29	2.02	96.00	1.86
sonar	86.83	9.38	86.83	9.38	90.73	4.36
splice	89.81	1.48	91.61	1.05	91.89	0.82
titanic	79.05	3.27	79.05	3.27	79.05	3.27
wdbc	94.16	1.94	94.16	1.94	95.58	1.40

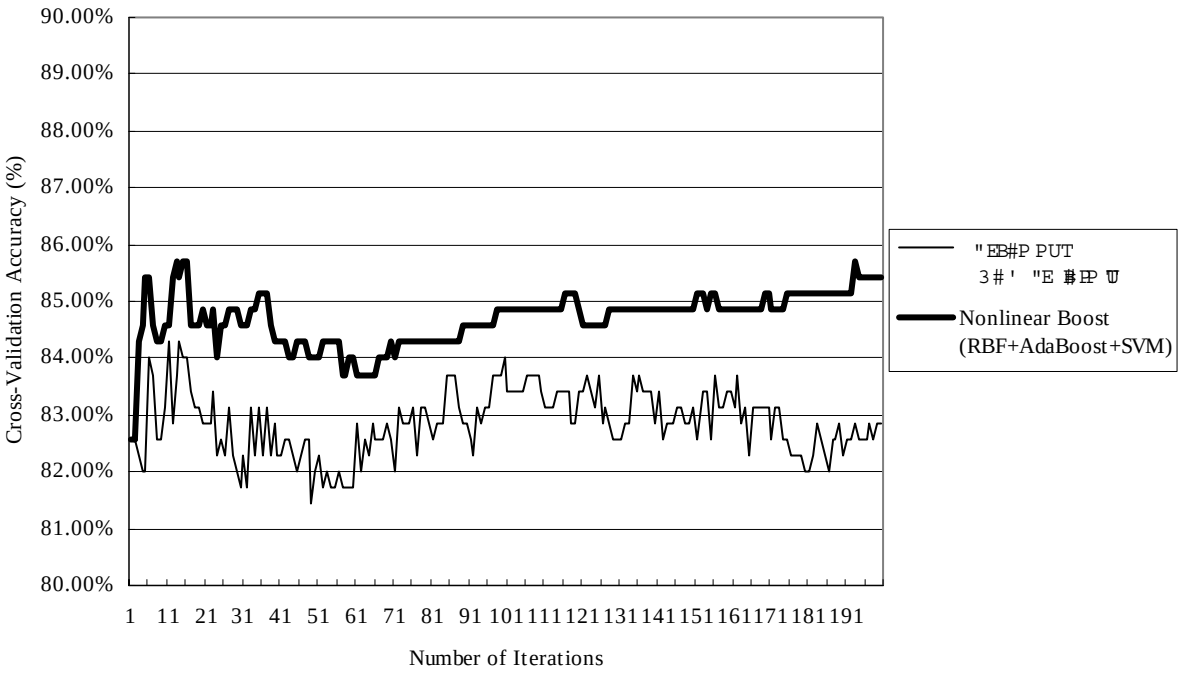

Figure 4. Cross-validation accuracies versus iteration times obtained when RBF+AdaBoost and RBF+AdaBoost+SVM were used for the ionosphere data set.