SPLBoost: An Improved Robust Boosting Algorithm Based on Self-Paced Learning

Kaidong Wang, Yao Wang, Qian Zhao, Deyu Meng, *Member, IEEE*, Xiuwu Liao, and Zongben Xu

***Abstract*—It is known that boosting can be interpreted as an optimization technique to minimize an underlying loss function. Specifically, the underlying loss being minimized by the traditional AdaBoost is the exponential loss, which proves to be very sensitive to random noise/outliers. Therefore, several boosting algorithms, e.g., LogitBoost and SavageBoost, have been proposed to improve the robustness of AdaBoost by replacing the exponential loss with some designed robust loss functions. In this article, we present a new way to robustify AdaBoost, that is, incorporating the robust learning idea of self-paced learning (SPL) into the boosting framework. Specifically, we design a new robust boosting algorithm based on the SPL regime, that is, SPLBoost, which can be easily implemented by slightly modifying off-the-shelf boosting packages. Extensive experiments and a theoretical characterization are also carried out to illustrate the merits of the proposed SPLBoost.**

***Index Terms*—AdaBoost, loss function, majorization minimization, robustness, self-paced learning (SPL).**

I. Introduction

For a classification or regression problem, there have been two natural ways to tackle it: one is that we can train a strong learning machine directly from the training set with a variety of machine learning methods, and expect that the obtained learning machine could have a satisfactory prediction accuracy; the other is that we can train a number of weak learners with slightly better accuracies than randomly guessing, then put them together in a specific way to get a strong learner that could have a better accuracy than those weak learners. The latter is the basic idea of ensemble learning. As an important and excellent ensemble learning framework, boosting [1]–[4] has been widely applied in many machine learning problems because of its simplicity and promising performance.

AdaBoost [5], [6] is one of the most commonly used boosting algorithms, and it has proven to be effective and easy to implement in various classification problems. Given training data $(x_1, y_1),\ldots,(x_n, y_n)$, where x_i is a vector-valued feature and $y_i \in \{1, -1\}$, it is known that AdaBoost can produce a strong learner $F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$, where $f_t(x)$ is the weak learner trained on weighted training data in the t-th step and α_t is a constant calculated based on the classification accuracy of $f_t(x)$. Then we can predict the label of a new sample by $\text{sign}(F(x))$. In particular, AdaBoost gives a weight initialized to $1/n$ to every training sample and adjusts it in every step such that the weights of the correctly classified samples by the current learner are decreased while the weights of misclassified samples are increased. This reweighting way gives rise to that AdaBoost always pays more attention to the samples which are hard to classify and ignores the easy-to-classify samples to some extent when training the next weak learner.

Many practical applications have demonstrated the success of AdaBoost in producing satisfactory and accurate strong classifiers, e.g., [7]–[10]. And it is interesting that in many cases, the test error seems to decrease consistently and then levels off rather than gradually increasing as more weak learners are added, which means that it is not prone to overfit and, as a result, it is not difficult for AdaBoost to determine the number of weak learners. Despite this, the classifiers produced by AdaBoost are not always acceptable, especially, when the training samples are corrupted by outliers [11]–[13]. It has been shown that the AdaBoost algorithm builds an additive logistic regression model for minimizing the expected risk based on the exponential loss function $\varphi(yf(x)) = \exp(-yf(x))$ [14]. It is easy to see that this loss increases rapidly with the increase of the magnitude of the negative margin $-yf(x)$, which means that it will significantly enlarge the functions of those large noises in training since they will have a large negative value of $yf(x)$. This naturally degenerates the performance of the approach in the presence of heavy noises/outliers.

Unfortunately, such biased training data with noisy labels are commonly encountered practically. A typical example is

Manuscript received October 3, 2018; revised January 25, 2019 and October 6, 2019; accepted November 18, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1402600, in part by the National Natural Science Foundation of China under Grant 11501440, Grant 11971374, Grant 61603292, Grant 61661166011, and Grant 91846110, in part by the Fundamental Research Funds for the Central Universities under Grant xjj2018085, in part by MoE-CMCC "Artificial Intelligence" Project under Grant MCM20190701, and in part by the China Postdoctoral Science Foundation under Grant 2017M610628 and Grant 2018T111031. This article was recommended by Associate Editor Y. Tan. *(Corresponding author: Yao Wang.)*

K. Wang is with the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an 710049, China, and also with the Center for Intelligent Decision-Making and Machine Learning, School of Management, Xi'an Jiaotong University, Xi'an 710049, China.

Y. Wang and X. Liao are with the Center for Intelligent Decision-Making and Machine Learning, School of Management, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: yao.s.wang@gmail.com).

Q. Zhao and Z. Xu is with the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an 710049, China.

D. Meng is with the Faculty of Information Technology, Macau University of Science and Technology, Macau 999078, China, and also with the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an 710049, China.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2019.2957101

a dataset roughly collected from a crowdsourcing system [15] or some search engines [16], [17], which would yield a large amount of noisy labels. This even makes human labelers be susceptible to errors in labeling. For instance, in the image classification application, certain image categories may be hard to discern. Thus, it is important to improve the robustness of AdaBoost to resist those noise lables/outliers in practice.

Aiming at remedying the poor robustness issue of AdaBoost, many studies have been conducted to improve its performance. The mainly adopted methodology is to design a new robust loss function to be optimized, and then use gradient descent-like strategies to resolve this new optimization problem. Such robust loss needs to be designed to increase evidently slower when the magnitude of $-yf(x)$ becomes larger, so as to suppress the effect of large noises and outliers. Although those robust boosting algorithms are proven to be able to have better performance than AdaBoost for dealing with outliers, there are natural defects for the idea to directly design and optimize new loss functions. Although easy to optimize, the convex loss functions are not robust enough to eliminate the impact of outliers. Though the nonconvex loss functions can often have better performances than the convex loss functions, using nonconvex loss could produce a nonconvex optimization problem which is usually difficult to get reliable and stable solutions. In this article, taking advantage of the robustness of the self-paced learning (SPL) regime, we come up with a new thought for robust boosting algorithms. Our main contributions can be summarized as follows.

First, we propose a new robust boosting algorithm named by SPLBoost, which incorporates the SPL into the AdaBoost framework. As mentioned above, it is not always a good idea to improve the robustness of AdaBoost by directly modifying the loss function, which motivates us to utilize another efficient way to achieve the same goal. It has been recently shown that SPL is an effective robust learning regime and has achieved satisfactory results in deal with many machine learning and computer version problems. A basic idea of SPL is to give weights in [0, 1] to training samples so that weights of the samples with larger losses are smaller and can then be zero when the corresponding losses are large enough. By combining SPL with AdaBoost, SPLBoost is proven to be able to improve the robustness of AdaBoost.

Second, the proposed SPLBoost algorithm can be very easily embedded into any off-the-shelf AdaBoost package. Besides, more SPLBoost variations can be easily designed by integrating it directly to other boosting algorithms, like LogitBoost and L_2Boost, to improve their robustness in the presence of heavy noises/outliers.

Third, we prove that SPLBoost exactly complies with the widely known majorization–minimization (MM) algorithm that is implemented by a latent objective function based on a nonconvex loss function. This clearly explains that why SPLBoost could be more robust than AdaBoost. In addition, by alternately optimizing two subproblems that are easy to solve rather than directly optimizing the latent objective function, SPLBoost can keep away from the annoying nonconvex optimization problem and get a better local solution.

Finally, the superiority of SPLBoost is extensively substantiated in the synthetic dataset, 17 UCI datasets and a real-world dataset, as compared with other state-of-the-art robust boosting algorithms.

The remainder of this article is organized as follows. We shall provide a brief review on boosting algorithms and SPL in Section II. Then, the SPLBoost algorithm and its theoretical analysis are presented in Section III. Section IV shows the experimental results on several synthetic, UCI, and real-world datasets. Several concluding remarks are finally made in Section V.

II. Related Work

A. Learning With Noisy Labels

Learning from data with noisy labels is an interesting problem both theoretically and practically. Bylander [18] concerned the learnability of linear threshold functions in the presence of classification noise, which is one of the earliest known attempts at learning with noisy labels. Lawrence and Schölkopf [19] proposed a Bayesian approach for constructing a kernel Fisher discriminant from training examples with noisy labels, which allows associating with each example a probability of the label being flipped. The work [20] studied the binary classification problem in the presence of random classification noise and provided two approaches to suitably modify any given surrogate loss function. Cesa-Bianchi *et al.* [21] focused on online learning of linear- and kernel-based predictors in the case where instances as well as labels are noisy.

Except for the traditional methods, dealing with noisy labels is also an active area in deep learning especially for learning deep convolutional neural networks (CNNs) [22]–[24]. Sukhbaatar *et al.* [22] proposed to introduce an extra noise layer into the CNN which adapts the network outputs to match the noisy label distribution and achieve the desired performance. Xiao *et al.* [23] proposed a general training framework that model the relationships between images, class labels, and label noises with a probabilistic graphical model and further integrate it into an end-to-end deep learning system to correct the noisy labels. Wang *et al.* [24] proposed an iterative learning framework for training CNNs on datasets with open-set noisy labels. This approach detects noisy labels and learns deep discriminative features in an iterative fashion and can robustly train CNNs in the presence of a high proportion of open-set as well as closed-set noisy labels.

B. Robust Boosting Algorithms

As aforementioned, the main defect of AdaBoost is that it can easily overfit to outliers, which inspires a lot of studies on improving the robustness of AdaBoost [25]–[27]. Generally speaking, there are three factors that affect the robustness of boosting algorithms: the loss function, the way to compute the weak learners, and the regularization term. Based on those factors, many robust boosting algorithms have been suggested.

Friedman *et al.* [14] proved that both discrete and real AdaBoost algorithms build an additive logistic regression model via Newton-like updates for optimizing the expected risk based on the exponential loss function $\varphi(yf(x)) =$

$\exp(-yf(x))$. With this, Friedman *et al.* [14] proposed two different robust boosting algorithms: 1) LogitBoost and 2) GentleBoost. The LogitBoost algorithm uses the Newton steps for optimizing the logistic loss $\varphi(yf(x)) = \log(1 + \exp(-yf(x)))$ which is more robust than the exponential loss. It is easy to see that the logistic loss function assigns fewer penalties to those samples with negative margins whose absolute values are very large, which are usually outliers. This is why LogitBoost is not so easy to overfit to the outliers. GentleBoost is the other robust boosting algorithm proposed, and it is different from the LogitBoost in the way of optimizing the underlying loss function. Basically, GentleBoost optimizes the exponential loss function as AdaBoost does. The main difference between GentleBoost and Real AdaBoost is that GentleBoost computes the weak learners by using an adaptive Newton step as LogitBoost does. For Real AdaBoost, the update $f_t(x)$ is half log-ratio which can be numerically unstable and lead to very large updates or sample weights. However, the updates of GentleBoost lie in the range $[-1, 1]$, leading to more conservative sample weights. Consequently, the influence that outliers exert on GentleBoost is weaker than AdaBoost.

As we could see, the loss functions of the aforementioned boosting algorithms are all convex. Theoretical properties of the boosting algorithms based on the convex loss functions have been extensively studied (see [28], [29]). The minimum of the convex loss function is easy to compute, which gives rise to the simplicity of those boosting algorithms. However, it has been shown that the convex loss function is not robust enough to tolerate noise and as a result, such boosting algorithms based on convex loss are naturally not insensitive to outliers. Specifically, Long and Servedio [30] proved that any boosting algorithm based on the convex loss functions is easily affected by random label noise and they present a sample example named Long/Servedio problem which cannot be learned by those popular boosting algorithms. As such, the results presented by Long and Servedio lead to a series of studies on boosting algorithms with nonconvex loss functions.

Based on the Boost-by-Majority algorithm [31] and BrownBoost [32], Freund [33] proposed a new robust boosting algorithm named RobustBoost which is more robust against outliers than AdaBoost and LogitBoost. The loss function of RobustBoost is nonconvex and changes during the boosting process. RobustBoost improves the robustness by allowing a preassigned error of margin maximization, and in each step, it updates and solves a differential equation and updates the preassigned target error.

It is known that most classifiers design algorithms to determine the optimal classifier through three steps: define a proper loss function $\phi(yf(x))$, determine a function class $\mathcal{F}$, and search within $\mathcal{F}$ for the function which optimizes the objective function based on a predefined loss function. In view of the limitations of these methods, such as low convergence rate and high sensitivity to the outliers, Masnadi-shirazi and Vasconcelos [34] showed that the problem of classifier design is identical to the problem of probability elicitation, which can be seen as a reverse procedure for solving the classification problem. This provides some new insights on the relationship between the loss function, the minimum risk, and the optimal classifier. With this, they derived a new loss function named Savage loss which trades convexity for boundedness. Using this new loss, they proposed so-called SavageBoost, that is, a new robust boosting algorithm which is more outlier resistant than AdaBoost and LogitBoost. Such kind of Savage loss is defined as $\phi(v) = (1/[(1 + e^{2v})^2])$, which clearly shows that unlike the exponential loss and logistic loss where the penalty always increases at a fast speed, Savage loss is nonconvex and quickly becomes constant as $v \longrightarrow -\infty$. Considering that the weights of the misclassified samples with large margins could be not large, SavageBoost is not sensitive to the outliers, as compared to AdaBoost and LogitBoost.

To further improve the robustness of SavageBoost, Miao *et al.* [35] proposed two robust boosting algorithms named RBoost1 and RBoost2, which have a deep connection with SavageBoost. Both the two boosting algorithms try to optimize the conditional expected risk based on the Savage2 loss function, that is, a new robust loss function is defined as $\phi(v) = (1/[(1 + e^{v})^2])$. It is easy to see that the Savage2 loss function is with the similar form to the Savage loss function and the only difference is the second-order factor in the denominator, which makes the Savage2 loss to give a gentler penalty for the misclassified samples with large margins than the Savage loss function. As such, the proposed RBoost could be more insensitive to the outliers than SavageBoost. In fact, two reasons weaken the robustness of SavageBoost: 1) the results that the weak learners in SavageBoost output are required to be the posterior probability estimation, and to estimate the posterior probability is more difficult than classification, and 2) the way SavageBoost computes the weak learners is not numerically stable. To avoid such two drawbacks of SavageBoost, the RBoost algorithms are carefully designed to optimize the Savage2 loss function. Precisely, the RBoost1 algorithm uses the adaptive Newton step to solve the minimization problem which computes a new weak learner based on the current classifier so that the conditional expected risk is maximally decreased. The RBoost2 algorithm is designed to make RBoost1 algorithm which restricts the weak learner algorithms to the regression methods adaptive to more general weak learner algorithms. With the use of more robust loss function and numerically stable methods to compute the weak learners, RBoost1 and RBoost2 algorithms could get good performance in dealing with noisy data.

As previously mentioned, most of the existing robust boosting algorithms try to design various robust loss functions with restricted penalties on misclassified samples with large margins and then design stable computation methods to compute the weak learners based on those loss functions. However, this common framework cannot always be satisfactory. Specifically, although nonconvex loss functions possess better antinoise ability than the convex loss functions, they induce nonconvex optimization problem to compute weak learners, which is very difficult task in general. In this article, instead of directly designing new robust loss functions, we propose a new robust boosting algorithm named SPLBoost by combining the classical Discrete AdaBoost algorithm with the robust learning idea of SPL.

It is worth mentioning that Pi *et al.* [36] proposed a similar classifier algorithm named self-paced boost learning (SPBL), which also combines the ideas of original boosting and SPL. However, there are significant differences between this algorithm and our proposed SPLBoost. First, SPBL is constructed via a max-margin view, while SPLBoost is constructed via a statistical view, that is, a stagewise additive modeling view. This makes the optimization procedures of such two algorithms are totally different. Precisely, for SPBL, the complicated column generation method is used, while for SPLBoost, based on our analysis presented later, we only need to slightly modify the original AdaBoost and thus makes it to be easily implemented by using the off-the-shelf packages. Second, we provide a theoretical analysis of SPLBoost in Section III-B, which clarifies why SPLBoost is capable of performing robustness in outlier/heavy noise cases, while there has not any similar theoretical results presented in [36]. Considering the above two reasons, the proposed SPLBoost is a totally different robust boosting algorithm.

C. Self-Paced Learning

Humans and animals often learn from the examples which are not randomly presented but organized in a meaningful order which gradually includes from easy and fewer concepts to complex and more ones. Inspired by this principle, Bengio *et al.* [37] proposed a new learning paradigm named curriculum learning which is the origin of SPL. In curriculum learning, a model is learned by gradually including from easy to complex samples in training to improve the accuracy of the model. Obviously, the key of curriculum learning is to find a proper ranking function which assigns learning priorities to training samples. To obtain a satisfactory model, the quality of the curriculum, that is, ranking function, is very important and it is oftentimes derived by predetermined heuristics for particular problems in real applications. This may lead to inconsistency between the fixed curriculum and the dynamically learned models.

To alleviate the aforementioned issue, Kumar *et al.* [38] proposed a new model named SPL in which instead of being derived by predetermined heuristics, the curriculum design is embedded as a regularization term into the learning objective. The formulation of SPL model is

$$\min_{\mathbf{w},\mathbf{v}\in[0,1]^n} \mathbf{E}(\mathbf{w},\mathbf{v};\lambda) = \sum_{i=1}^{n} v_i L(y_i, g(\mathbf{x}_i,\mathbf{w})) - \lambda \sum_{i=1}^{n} v_i \quad (1)$$

where λ is an age parameter for controlling the learning pace, and $L(y_i, g(\mathbf{x}_i,\mathbf{w}))$ denotes the loss function which calculates the cost between the ground-truth label y_i and the estimated label $g(\mathbf{x}_i,\mathbf{w})$. Here, $\mathbf{w}$ denotes the model parameter inside the decision function g, and $\mathbf{v} = [v_1,\ldots,v_n]^T$ denotes the latent weight variable which is induced by a function $f(v_i;\lambda)$). Thus, (1) can be expressed as

$$\min_{\mathbf{w},\mathbf{v}\in[0,1]^n} \mathbf{E}(\mathbf{w},\mathbf{v};\lambda) = \sum_{i=1}^{n}(v_i L(y_i, g(\mathbf{x}_i,\mathbf{w})) + f(v_i;\lambda)) \quad (2)$$

where $f(v_i;\lambda)$ represents the so-called self-paced regularizer, whose intrinsic conditions have been theoretically abstracted in [39] and [40]. Now, the objective of SPL is to minimize the weighted training loss together with a self-paced regularizer (SP-regularizer). Basically, (2) can be generally solved by the alternative search strategy (ASS) method, in which the variables are divided into two disjoint blocks and in each iteration, a block of variables is optimized while keeping the other block fixed. More precisely, with the fixed $\mathbf{v}$, (2) is a weighted training loss minimization problem which appears in many machine learning problems. And with the fixed $\mathbf{w}$, (2) is a common regularization problem about $\mathbf{v}$ which can be easily solved. For example, following [38], if $f(v_i;\lambda)$ is chosen as $f(v_i;\lambda) = -(1/\lambda)\|\mathbf{v}\|_1$, then the global optimum $\mathbf{v}^* = [v_1^*,\ldots,v_n^*]^T$ is easily calculated by

$$v_i^* = \begin{cases} 1, & L(y_i, g(\mathbf{x}_i,\mathbf{w})) < \lambda \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

It is not hard to see that the aforementioned SPL model implements the automatic selection of samples and then trains model only on those selected samples. When we update $\mathbf{v}$ with a fixed $\mathbf{w}$, samples with losses which are smaller than a certain threshold λ are considered to be "easy" samples and will be selected in training (i.e., $v_i^* = 1$), while the rest are considered to be too "difficult" to be learned for this λ and will be abandoned (i.e., $v_i^* = 0$). When we update $\mathbf{w}$ with a fixed $\mathbf{v}$, the classifier is trained only on the selected easy samples. The parameter λ corresponds to the "age" of the model that determines the ability of the model to learn difficult samples. When λ is small, the model can only learn from easy samples with small losses and as λ grows, more samples with larger losses can be learned to train a more "mature" model.

The SP-regularizer chosen as the negative ℓ_1-norm induces the variable $\mathbf{v}$ which takes only binary values, that is, $v_i = 1$ (selected samples) and $v_i = 0$ (unselected samples). This scheme is called hard weighting. Hard weighting can only determine whether a sample should be selected, which is not good enough because sometimes we also want to discriminate the importance of samples. Jiang *et al.* [39] theoretically abstracted the intrinsic conditions of the SP-regularizer and proposed several soft weighting schemes, such as linear soft weighting and logarithmic soft weighting. Zhao *et al.* [40] used SPL for matrix factorization and proposed a new soft weighting scheme named mixture weighting which is a hybrid of the soft and the hard weighting. Li *et al.* [41] proposed a novel polynomial soft weighting regularizer with an adjustable parameter in their multiobjective SPL model. Soft weighting assigns the real-valued weights that reflects the latent importance of samples in training more faithfully, which is more reasonable and general than hard weighting. In the following text, we shall list the formulations of linear soft weighting, mixture weighting, and polynomial soft weighting, respectively, together with their closed-form solutions $v^*(\lambda;\ell)$.

Linear Soft Weighting:

$$f^L(v;\lambda) = \lambda\left(\frac{1}{2}v^2 - v\right);\ v^*(\lambda;\ell) = \begin{cases} -\ell/\lambda + 1, & \text{if } \ell < \lambda \\ 0, & \text{if } \ell \geq \lambda. \end{cases}$$

Mixture Weighting:

$$f^M(v;\lambda,\gamma) = \frac{\gamma^2}{v + \gamma/\lambda}$$

$$v^*(\lambda, \gamma; \ell) = \begin{cases} 1, & \text{if } \ell \leq \left(\frac{\lambda\gamma}{\lambda+\gamma}\right)^2 \\ 0, & \text{if } \ell \geq \lambda^2 \\ \gamma\left(\frac{1}{\sqrt{\ell}} - \frac{1}{\lambda}\right), & \text{otherwise.} \end{cases}$$

Polynomial Soft Weighting:

$$f^P(v; \lambda, t) = \lambda\left(\frac{1}{t}||v||_2^t - \sum_{i=1}^n v_i\right)$$

$$v^*(\lambda, t; \ell) = \begin{cases} (1 - \ell/\lambda)^{1/(t-1)}, & \text{if } \ell < \lambda \\ 0, & \text{if } \ell \geq \lambda. \end{cases}$$

on (1), many variations of the SPL regime have been proposed, such as self-paced reranking [39], SPL with diversity [42], self-paced curriculum learning [43], and self-paced multiple-instance-learning [44]. And applications of SPL in many machine learning and computer version tasks, such as objective detector adaptation [45], long-term tricking [46], face identification [47], and specific-class segmentation learning [48], have shown its effectiveness especially its robustness when dealing with severally corrupted data.

Meng and Zhao [49] provided some theoretical evidences to illustrate the insights under SPL. They proved that the ASS algorithm to solve the SPL problem exactly accords with the MM algorithm implemented on a latent nonconvex SPL objective function. Their work laid the theoretical foundation for SPL.

Comparing SPL with AdaBoost, we can see that there is one thing in common between them, that is, training samples with different losses are unequal and will be endowed with different weights. The way of SPL and AdaBoost to assign weights to samples is different. For AdaBoost, samples with large losses are paid more attention to and are given larger weights. On the contrary, for SPL, samples with losses larger than a certain constant are thought as outliers and their weights are zero. On account of the fact that the reason why AdaBoost is not robust is that AdaBoost assigns too large weights to the samples with very large losses and those samples are usually outliers, we expect that SPL provides a complementary assistance to restrict the sample weights of AdaBoost and improve its robustness. Therefore, a new robust boosting algorithm named SPLBoost is proposed based on this idea.

III. SPLBoost

A. Algorithm

Given training data $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i is a vector-valued feature and $y_i \in \{1, -1\}$. It is known that AdaBoost algorithm builds an additive logistic regression model for minimizing the expected risk based on the exponential loss function $\varphi(yf(x)) = \exp(-yf(x))$. In each iteration, supposing that we have a current classifier $F(x)$, AdaBoost then seeks a new weak learner $f(x)$ through the following optimization problem:

$$\{\alpha, f\} = \arg\min_{\alpha, f} \sum_{i=1}^n e^{-y_i(F(x_i)+\alpha f(x_i))}. \tag{4}$$

Embedding (4) in a general SPL model, we can get a new algorithm named SPLBoost, which seeks a new weak learner $f(x)$ and updates its weight α in the final strong learner in every step

$$\{\alpha, f, \mathbf{v}\} = \arg\min_{\alpha, f, \mathbf{v}} \sum_{i=1}^n v_i e^{-y_i(F(x_i)+\alpha fl(x_i))} + \hat{f}(v_i, \lambda) \tag{5}$$

where $\hat{f}$ is an SP-regularizer and $\mathbf{v} = [v_1, \ldots, v_n]^T$ is the latent weight variable induced from $\hat{f}$, and λ is the age parameter. It is easy to see that the objective (5) of SPLBoost in each step is to minimize a weighted exponential loss together with a self-paced regularizer. In AdaBoost, the exponential loss is directly minimized and the outliers whose losses are usually very large are easy to be paid more attention to. SPLBoost overcomes this problem by assigning different weights v to the exponential losses of training samples. Although different SP-regularizer can induce different format of v, v will always be zero when the loss is very large, which means that the corresponding sample is very likely to be outlier. Thus, SPLBoost can eliminate the negative influence of the outliers in training data to a large extent and improve the robustness of AdaBoost.

As in SPL, we shall solve (5) by the ASS method. In order to distinguish the iterative process of the ASS from the iterative process of SPLBoost for updating classifier $F(x)$, we call the former inner iteration, and the latter outer iteration. In every inner iteration, with fixed α and f, (5) is a optimization problem about $\mathbf{v}$ with global optimum $\mathbf{v}^* = [v_1^*, \ldots, v_n^*]^T$ whose form is different for different SP-regularizer and has been presented in relevant papers [38], [39], [41]. Especially, when the SP-regularizer is the negative ℓ_1-norm, or say, hard weighting, we can plug the above exponential loss into (3) and then $\mathbf{v}^*$ can be easily calculated by the following formulation:

$$v_i^* = \begin{cases} 1, & e^{-y_i(F(x_i)+\alpha f(x_i))} < \lambda \\ 0, & \text{otherwise.} \end{cases}$$

With fixed $\mathbf{v}$, (5) is a weighted exponential loss minimization problem

$$\{\alpha, f\} = \arg\min_{\alpha, f} \sum_{i=1}^n v_i e^{-y_i(F(x_i)+\alpha f(x_i))}. \tag{6}$$

Equation (6) is similar to the minimization problem (4), thus we can solve it by using the similar idea of AdaBoost. Follow the way in [14], for fixed α, we expand (6) to second order about $f(x_i) = 0$

$$f = \arg\min_f \sum_{i=1}^n v_i e^{-y_i(F(x_i)+\alpha f(x_i))}$$
$$\approx \arg\min_f \sum_{i=1}^n v_i e^{-y_i(F(x_i))}\Big(1 - y_i\alpha f(x_i) + \alpha^2$$

Taking $f(x_i) \in \{-1, 1\}$ and $y_i \in \{-1, 1\}$ into consideration, we have

$$f = \arg\min_f \sum_{i=1}^n v_i e^{-y_i(F(x_i))}\left(2 - 2y_i\alpha f(x_i) + \alpha^2 f(x_i)^2\right)$$
$$= \arg\min_f \sum_{i=1}^n v_i e^{-y_i(F(x_i))}\left(1 - 2y_i\alpha f(x_i) + \alpha^2 f(x_i)^2\right)$$

$$= \arg\min_f \sum_{i=1}^{n} v_i e^{-y_i(F(x_i))}\left(y_i^2 - 2y_i\alpha f(x_i) + \alpha^2 f(x_i)^2\right)$$
$$= \arg\min_f \sum_{i=1}^{n} v_i e^{-y_i(F(x_i))}(y_i - \alpha f(x_i))^2$$

where we denote $w_i = e^{-y_i(F(x_i))}$ which is the same as the sample weights defined in AdaBoost. We can find that solving the above-weighted least-squares problem to obtain the weak learner $f(x)$ is equivalent to implement AdaBoost except for the use of latent weight variable $\mathbf{v}$. Thus, imitating the approach in AdaBoost, one can train $f(x)$ from the training data with sample weights $v_i w_i$ rather than w_i. Given $f(x) \in \{-1, 1\}$, we can directly optimize (6) to determine α

$$\alpha = \arg\min_\alpha \sum_{i=1}^{n} v_i w_i e^{-y_i \alpha f(x_i)}$$
$$= \arg\min_\alpha \sum_{y_i = f(x_i)} v_i w_i e^{-\alpha} + \sum_{y_i \neq f(x_i)} v_i w_i e^{\alpha}. \quad (7)$$

It is not hard to see the above objective is a convex function, thus to get the optimal α, we can directly calculate its derivative and set it to be zero

$$-\sum_{y_i = f(x_i)} v_i w_i e^{-\alpha} + \sum_{y_i \neq f(x_i)} v_i w_i e^{\alpha} = 0. \quad (8)$$

From (8), we have

$$\alpha = \frac{1}{2}\log \frac{\sum_{y_i = f(x_i)} v_i w_i}{\sum_{y_i \neq f(x_i)} v_i w_i}$$
$$= \frac{1}{2}\log \frac{1 - \sum_{y_i \neq f(x_i)} v_i w_i}{\sum_{y_i \neq f(x_i)} v_i w_i}$$
$$= \frac{1}{2}\log \frac{1 - \text{err}}{\text{err}} \quad (9)$$

where $\text{err} = \sum_{y_i \neq f(x_i)} v_i w_i$ is the weighted misclassification error of weak learner $f(x)$. It is easy to see that the formula of α in (9) is also consistent with AdaBoost except for the latent weight variable $\mathbf{v}$.

By alternative iteration, we can calculate the optimal $\mathbf{v}$, α, and f for (5). Then, we can obtain the update for $F(x)$

$$F(x) \leftarrow F(x) + \alpha f(x). \quad (10)$$

In the next outer iteration, the latent weight variable $\mathbf{v}$ can be initialized to the current $\mathbf{v}$, and w_i is updated as follows:

$$w_i \leftarrow w_i e^{-\alpha y_i f(x_i)}. \quad (11)$$

Since $-y_i f(x_i) = 2 \times 1_{y_i \neq f(x_i)} - 1$, the update is equivalent to

$$w_i \leftarrow w_i \exp\left(\log \frac{1 - \text{err}}{\text{err}} 1_{y_i \neq f(x_i)}\right). \quad (12)$$

This update of w is clearly the same as that in AdaBoost.

The details of SPLBoost are summarized in Algorithm 1. Next, we shall give some remarks about it.

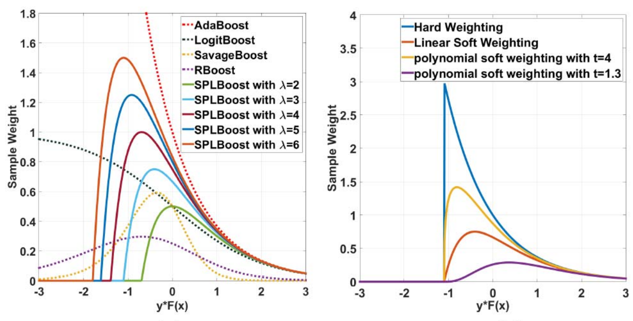
Remark 1: There are two layers of iteration in Algorithm 1, that is, the outer iteration is to update the classifier $F(x)$, and the inner iteration is to find the optimal latent weight variable $\mathbf{v}$, the weak learner $f(x)$ and its weight α in the current outer iteration. When a new inner iteration starts, the latent weight variable $\mathbf{v}$ is initialized to the optimal value provide by the last outer iteration and our experiments show that in this case, the inner iteration is rapidly converge and there have no significant difference between the case where the inner iteration runs only one step and the case where the inner iteration keeps running until converged. As such, we actually set the inner iteration as one step to speedup the algorithm implementation.

Algorithm 1 SPLBoost Algorithm

```
Input: training samples $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, iteration count $T$, parameter $\lambda$;
Initialization: $w_i = 1/n$, $v_i = 1$, $i = 1, \cdots, n$;
  for t=1 to $T$ do
    while not converge do
      1. Fit the classifier $f_t(x) \in \{-1, 1\}$ using weights $v_i w_i / (\sum_i v_i w_i)$ on the training data;
      2. Compute err $= \sum_{y_i \neq f(x_i)} v_i w_i / \sum_i v_i w_i$, $\alpha_t = \frac{1}{2}\log\frac{1-\text{err}}{\text{err}}$;
      3. Compute $\mathbf{v}$;
    end while
    4. Set $w_i \leftarrow w_i \exp\left(\log\frac{1-\text{err}}{\text{err}} 1_{y_i \neq f(x_i)}\right)$;
  end for
Output: The strong classifier $sign \sum_{t=1}^{T} \alpha_t f_t(x)$;
```

Remark 2: For the age parameter λ, it is easy to see that when the losses of samples are larger than λ, the latent weight variable of those samples could be zero, which means that those samples would not be selected during the training process. Thus, λ actually represents the "tolerance" of the algorithm toward noises and outliers. The larger λ is, the more "tolerant" the algorithm is to noises and outliers and the less the samples which are considered to be outliers that would be abandoned ($v_i = 0$). Furthermore, when λ is large enough, the SPLBoost algorithm degenerates into AdaBoost. On the contrary, the smaller λ is, the more "rigorous" the algorithm is to large noises and outliers and more samples would be abandoned. Apparently, the value of λ has a huge influence on the performance of the algorithm and thus it is important to select an appropriate λ. In practice, we usually select the proper λ via cross-validation.

Remark 3: Since that the weak learner $f(x)$ produced by Algorithm 1 is restricted in $\{-1, 1\}$, the losses of the samples can only be $e^{-\alpha}$ or e^{α} when we calculate $\mathbf{v}$ in the first outer iteration step, where α is the weight of the first weak learner. Considering that $v_i = 0$ for the samples whose losses are larger than λ, the samples that are misclassified by the first weak learner could all not be selected for training in the next outer iteration if λ falls between $[e^{-\alpha}, e^{\alpha}]$, which usually means that too many samples would be abandoned due to the low accuracy of the weak learner. To avoid this kind of unreasonable situation, one can adopt a warm start procedure, that is, in the first few outer iteration steps, set λ be a very large number instead of the input value, and after obtaining the corresponding weak learners, λ would be reset to be the input value. As such, the first few weak learners are trained by AdaBoost and as a result, the samples are determined whether



(a) (b)

Fig. 1. (a) Sample weight of various boosting algorithms, including AdaBoost, LogitBoost, SavageBoost, RBoost, and SPLBoost with $\lambda = 2, 3, 4, 5, 6$, where the SP-regularizer is linear soft weighting. (b) Sample weight of SPLBoost with various SP-regularizers, including hard weighting, linear soft weighting, and polynomial soft weighting with $t = 4$ and $t = 1.3$, where λ is fixed as 3.

they should be selected or not based on a classifier that is not so bad. According to our experience, it is reasonable that in the first three outer iterations λ is set to be 10^6, and then the satisfactory λ can be tuned in $[1.0, 6.0]$ by cross-validation.

Different from AdaBoost, where the sample weight is $w_i = e^{-y_i(F(x_i))}$, SPLBoost modifies the sample weight to be $v_i w_i$ by introducing the latent weight variable $\mathbf{v}$. We shall show that this is the reason why SPLBoost is more robust. Fig. 1(a) illustrates the sample weight of different boosting algorithms, including AdaBoost, LogitBoost, SavageBoost, RBoost, and SPLBoost with $\lambda = 2, 3, 4, 5, 6$, where the SP-regularizer is linear soft weighting. It is easy to observe that AdaBoost pays too much attention on the misclassified samples with very large margins which are usually outliers. Thus, AdaBoost is usually very sensitive to outliers. In LogitBoost, the weights of the misclassified samples with very large margins are smaller than that in AdaBoost, but they are still larger than the weights of any other samples, thus LogitBoost is still easily affected by outliers. For two popular robust boosting algorithms with nonconvex loss functions, that is, SavageBoost and RBoost, they give small weights to the misclassified samples with very large margins, which makes them insensitive to outliers. For SPLBoost, if the margin of the misclassified samples is larger than a certain constant which is determined by λ, the weights of those samples could be zero. In that way, SPLBoost can more thoroughly eliminate the influence of the misclassified samples, which are the outliers. Fig. 1(b) illustrates the sample weight of SPLBoost with different SP-regularizers, here λ is fixed as 3. One can see that different SP-regularizers provide different curves of the sample weight, which may be more suitable to deal with different data, and although those curves have various shapes, they could all be zero when the margins of the misclassified samples are large enough. This guarantees the robustness of all such SP-regularizers to outliers.

B. Theoretical Analysis

In this section, we shall provide some theoretical analysis of SPLBoost which shows a clear theoretical evidence to clarify why SPLBoost is capable of performing robustness especially in outlier/heavy noise cases. For convenience, we briefly write the exponential loss $e^{-y_i(F(x_i)+\alpha f(x_i))}$ as $\ell_i(\alpha, f)/\ell_i$ and $e^{-y(F(x)+\alpha f(x))}$ as $\ell(\alpha, f)/\ell$ in the following.

According to [49, Th. 1], it can be derived that, for latent weight variable $v^*(\lambda; l)$ conducted by an SP-regularizer and $\tilde{F}_\lambda(\ell)$ calculated by $\tilde{F}_\lambda(\ell) = \int_0^\ell v^*(\lambda; l)\, \mathrm{d}l$, and given a fixed α^* and f^*, it holds that

$$\tilde{F}_\lambda(\ell(\alpha, f)) \le Q_\lambda(\alpha, f|\alpha^*, f^*) \tag{13}$$

where

$$\begin{aligned} Q_\lambda(\alpha, f|\alpha^*, f^*) = {} & \tilde{F}_\lambda(\ell(\alpha^*, f^*)) \\ & + v^*(\lambda; \ell(\alpha^*, f^*))(\ell(\alpha, f) - \ell(\alpha^*, f^*)). \end{aligned}$$

With this, denote

$$\begin{aligned} Q_\lambda^{(i)}(\alpha, f|\alpha^*, f^*) = {} & \tilde{F}_\lambda(\ell_i(\alpha^*, f^*)) \\ & + v^*(\lambda; \ell_i(\alpha^*, f^*))(\ell_i(\alpha, f) - \ell_i(\alpha^*, f^*)) \end{aligned}$$

and we can then easily obtain that

$$\sum_{i=1}^{n} \tilde{F}_\lambda(\ell_i(\alpha^*, f^*)) \le \sum_{i=1}^{n} Q_\lambda^{(i)}(\alpha, f|\alpha^*, f^*) \tag{14}$$

which verifies that $\sum_{i=1}^n Q_\lambda^{(i)}(\alpha, f|\alpha^*, f^*)$ can be used as a surrogate function of $\sum_{i=1}^n \tilde{F}_\lambda(\ell_i(\alpha^*, f^*))$ in the MM algorithm. We can then ready to present the following result.

Theorem 1: The SPLBoost algorithm is equivalent to the MM algorithm on a minimization problem of the latent SPLBoost objective $\sum_{i=1}^n \tilde{F}_\lambda(e^{-y_i F(x_i)})$ with the latent loss $\tilde{F}_\lambda(e^{-yF(x)})$.

Proof: Assume that we have finished $(t-1)$ times outer iteration and got the classifier F_{t-1}. Denote f_t^k and α_t^k as the weak classifier and its weight got from the kth inner iteration in the tth outer iteration, then such two alternative search steps in the next iteration can be explained as a standard MM scheme. Precisely, there are two cases should be considered.

Case 1: If the inner iteration has not converged after the kth step, we denote the surrogate function

$$\begin{aligned} Q_\lambda^{(i)}\left(\alpha, f|\alpha_t^k, f_t^k\right) = {} & \tilde{F}_\lambda\left(\ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) + v^*\left(\lambda; \ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \\ & \times \left(\ell_i^{t-1}(\alpha, f) - \ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \end{aligned}$$

where $\ell_i^{t-1}(\alpha, f) = e^{-y_i(F_{t-1}(x_i)+\alpha f(x_i))}$.

Majorization Step: To obtain each $Q_\lambda^{(i)}(\alpha, f|\alpha_t^k, f_t^k)$, we only need to calculate $v^*(\lambda; \ell_i^{t-1}(\alpha_t^k, f_t^k))$ by solving the following problem under the corresponding SP-regularizer $\hat{f}(v_i, \lambda)$:

$$v^*(\lambda; \ell_i^{t-1}(\alpha_t^k, f_t^k)) = \arg\min_{v_i \in [0,1]} v_i \ell_i^{t-1}(\alpha_t^k, f_t^k) + \hat{f}(v_i, \lambda).$$

This exactly complies with updating $\mathbf{v}$ in Algorithm 1.

Minimization Step: we need to calculate

$$\begin{aligned} \left[\alpha_t^{k+1}, f_t^{k+1}\right] &= \arg\min_{\alpha, f} \sum_{i=1}^{n} \tilde{F}_\lambda\left(\ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \\ &\quad + v^*\left(\lambda; \ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \\ &\quad \times \left(\ell_i^{t-1}(\alpha, f) - \ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \\ &= \arg\min_{\alpha, f} \sum_{i=1}^{n} v^*\left(\lambda; \ell_i^{t-1}\left(\alpha_t^k, f_t^k\right)\right) \ell_i^{t-1}(\alpha, f) \end{aligned}$$

which is equivalent to update α and f in Algorithm 1.

Case 2: If the inner iteration has converged after the kth step with the finally learned weak classifier f_t and its weight α_t, we denote $F_t = F_{t-1} + \alpha_t f_t$ and the surrogate function

$$\begin{aligned} Q_\lambda^{(i)}(\alpha, f|\alpha_t, f_t) &= \tilde{F}_\lambda\left(\ell_i^{t-1}(\alpha_t, f_t)\right) + v^*\left(\lambda; \ell_i^{t-1}(\alpha_t, f_t)\right) \\ &\quad \times \left(\ell_i^{t-1}(\alpha, f) - \ell_i^{t-1}(\alpha_t, f_t)\right) \\ &= \tilde{F}_\lambda\left(\ell_i^t(\alpha = 0, f)\right) + v^*\left(\lambda; \ell_i^t(\alpha = 0, f)\right) \\ &\quad \times \left(\ell_i^t(\alpha, f) - \ell_i^t(\alpha = 0, f)\right) \end{aligned}$$

where $\ell_i^t(\alpha, f) = e^{-y_i(F_t(x_i) + \alpha f(x_i))}$.

Majorization Step: To obtain each $Q_\lambda^{(i)}(\alpha, f|\alpha_t, f_t)$, we only need to calculate $v^*(\lambda; \ell_i^t(\alpha = 0, f)) = v^*(\lambda; \ell_i^{t-1}(\alpha_t, f_t))$ by solving the following problem under the corresponding SP-regularizer $\hat{f}(v_i, \lambda)$:

$$v^*\left(\lambda; \ell_i^{t-1}(\alpha_t, f_t)\right) = \arg\min_{v_i \in [0,1]} v_i \ell_i^{t-1}(\alpha_t, f_t) + \hat{f}(v_i, \lambda).$$

This exactly complies with updating $\mathbf{v}$ in Algorithm 1.

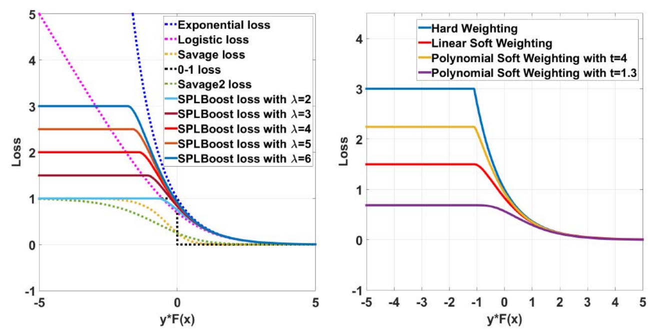
Minimization Step: we need to calculate

$$\begin{aligned} \left[\alpha_{t+1}^1, f_{t+1}^1\right] &= \arg\min_{\alpha, f} \sum_{i=1}^n \tilde{F}_\lambda\left(\ell_i^t(\alpha = 0, f)\right) \\ &\quad + v^*\left(\lambda; \ell_i^t(\alpha = 0, f)\right) \\ &\quad \times \left(\ell_i^t(\alpha, f) - \ell_i^t(\alpha = 0, f)\right) \\ &= \arg\min_{\alpha, f} \sum_{i=1}^n v^*\left(\lambda; \ell_i^t(\alpha = 0, f)\right) \ell_i^t(\alpha, f) \\ &= \arg\min_{\alpha, f} \sum_{i=1}^n v^*\left(\lambda; \ell_i^{t-1}(\alpha_t, f_t)\right) \ell_i^t(\alpha, f) \end{aligned}$$

which is equivalent to update α and f in Algorithm 1. ■

With Theorem 1, various off-the-shelf theoretical results of the MM algorithm can then be used to explain the properties of SPLBoost. Particularly, based on the well-known convergence theory of the MM algorithm, that is, the lower-bounded latent SPLBoost objective is monotonically decreasing during the iteration. Thus, a weak convergence result of SPLBoost can be directly obtained.

A number of formulas of the latent loss $\tilde{F}_\lambda(\ell) = \int_0^\ell v^*(\lambda; l)\, \mathrm{d}l$ under various SP-regularizers have been calculated and presented in [41] and [49], and we only need to plug the exponential loss function $\ell = \exp(-yF(x))$ into those formulas to get the latent SPLBoost losses under various SP-regularizers. Fig. 2(a) illustrates some popular loss functions, including exponential loss, logistic loss, Savage loss, Savage2 loss, 0-1 loss, and latent SPLBoost loss with $\lambda = 2, 3, 4, 5, 6$, where the SP-regularizer is linear soft weighting. It is easy to see from Fig. 2(a) that, compared with the original exponential loss function, the latent SPLBoost loss has an evident suppressing effect on the large losses. When the loss is larger than a certain threshold which is determined by the age parameter λ, the latent SPLBoost loss $\tilde{F}_\lambda(e^{-yF(x)})$ becomes a constant thereafter, which rationally explains why SPLBoost shows good robustness to the outliers and heavy noises. The misclassified samples with very large margins will have constant SPLBoost losses and thus have no effect on the model training due to their zero gradients. Corresponding to the original



(a) (b)

Fig. 2. (a) Various loss functions, including exponential loss, logistic loss, Savage loss, Savage2 loss, 0-1 loss, and latent SPLBoost loss with $\lambda = 2, 3, 4, 5, 6$, where the SP-regularizer is linear soft weighting. (b) Latent SPLBoost loss under various SP-regularizers, including hard weighting, linear soft weighting, and polynomial soft weighting with $t = 4$ and $t = 1.3$, where λ is fixed as 3.

SPLBoost model, the latent weight variable v_i of those large-loss samples will be 0, and thus those samples will have no influence on the training of the weak learners. λ actually determines the "degree" of the suppressing effect SPLBoost loss has on the large losses. The larger λ is, the gentler the suppressing effect is, and vice-versa. When $\lambda = \infty$, the suppressing effect completely disappears and the latent SPLBoost loss under hard weighting SP-regularizer degenerates into an exponential loss. Fig. 2(b) illustrates the latent SPLBoost loss under various SP-regularizers, including hard weighting, linear soft weighting, and polynomial soft weighting with $t = 4$ and $t = 1.3$, where λ is fixed as 3. We can see that different SP-regularizers give different shapes of the latent SPLBoost loss, but they tend to become constant when the loss is larger than a certain constant, which guarantees their robustness to the outliers and heavy noises.

According to the above analysis, it is easy to see that SPLBoost is actually an optimization model with a nonconvex loss function. Different from many other robust boosting algorithms which directly optimize the nonconvex objective functions, SPLBoost decomposes the minimization of the robust but difficult-to-solve nonconvex problem into two much easier optimization problems with respect to the latent weight variable $\mathbf{v}$ and the weak learner $f(x)$. In this sense, SPLBoost remedies the difficulty of nonconvex optimization and simplifies the solving process.

IV. Experiments

In this section, we shall test the robustness of the proposed SPLBoost algorithm through a series of experiments.

A. Synthetic Gaussian Dataset

It has been shown that the loss functions used by the boosting algorithms have an important effect on the robustness to the heavy noise/outliers. Moreover, the reweighting strategy of a boosting algorithm directly comes from the loss function it used and determines how much attention the algorithm pays to various of samples. Thus, a reasonable reweighting strategy is necessary to a robust boosting algorithm and to weaken the interference of the outliers to the training, a good reweighting strategy should give the least possible weights to the outliers. In the synthetic dataset, the underlying distribution of the samples and the information of the outliers added to those samples are known, thus it is easy to determine the optimal Bayes decision boundary for the classification problem and observe the rationality of the distribution of the sample weights. To compare the reweighting strategies of different boosting algorithms, we first evaluate the proposed SPLBoost algorithm, AdaBoost and some other robust boosting algorithm, including LogitBoost, SavageBoost, RBoost, and RobustBoost, on a synthetic Gaussian dataset.

According to the 2-D Gaussian distributions

$$N\left([2, -2], \begin{bmatrix} 2.5 & 1.5 \\ 1.5 & 5 \end{bmatrix}\right)$$

and

$$N\left([-2, 2], \begin{bmatrix} 2.3 & -0.7 \\ -0.7 & 2.3 \end{bmatrix}\right)$$

we first generate 100 samples for both the negative and positive classes, then randomly select 15% from both the two classes and reverse their labels, and those samples selected can be considered to be outliers. In this way, we obtain the two-class training data with 15% outliers and then train the classifiers using the aforementioned boosting algorithms. For AdaBoost, SPLBoost, and RobustBoost, the classification tree C4.5 is selected to be the weak learner and in SPLBoost, the hard weighting is used to be the SP-regularizer. And for LogitBoost and RBoost, the regression tree CART is used to be the weak learner. As for the parameters in those algorithms, the maximum iteration step is set to be 200, and a five-fold cross-validation procedure is used to determine the appropriate number of iteration steps and other parameters.

Fig. 3 illustrates the performance of all the competing boosting algorithms. In all the subfigures of Fig. 3, pluses represent the positive training samples and cross marks represent the negative training samples. Blue squares represent those training samples that are generated from the Gaussian distribution of the negative class, but are labeled as positive samples. They are actually the outliers added to the negative class. In a similar way, the red circles represent the outliers that added to the positive class. The black circles represent the training samples whose sample weights are 0 in SPLBoost, that is, the samples which are considered to be outliers and thus have no effects on the weak learner training in the SPLBoost algorithm. To visually observe the distribution of the sample weights of the various boosting algorithms, the sizes of the pluses, cross marks, blue squares, and red circles are in proportion to the sample weights of the corresponding training samples. Since that the weights of the training samples marked by the black circles are 0, the sizes of those black circles are all the same and have no relationship with their weights.

Basically, several observations can be made from Fig. 3. First, in Fig. 3(b), most of the points with large sizes are surrounded by the red circles or blue squares, which means that most of the training samples with large sample weights are outliers. This reveals that AdaBoost is so sensitive to the outliers. Actually, the exponential loss used by AdaBoost makes

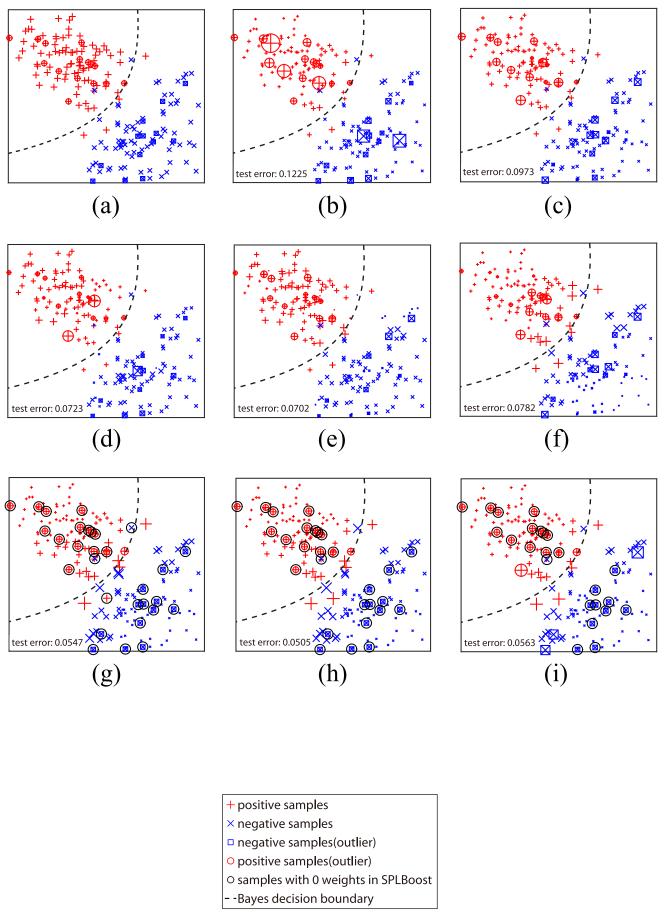


Fig. 3. Optimal Bayes decision boundary and the sample weights of AdaBoost, LogitBoost, SavageBoost, RBoost, RobustBoost, and SPLBoost with $\lambda = 1.5, 2.0, 2.5$, where the SP-regularizer in SPLBoost is hard weighting.

it to pay too much attention to the misclassified samples with large margins. Second, it can be seen from Fig. 3(c) that the weights of most of the outliers are still larger than that of the correct training samples, though LogitBoost gives gentler weight on such outliers than AdaBoost. This is a common drawback of the convex loss functions, as stated by [30]. Third, Fig. 3(d)–(f) shows the performance of SavageBoost, RBoost and RobustBoost, respectively, which are more satisfactory than that of AdaBoost and LogitBoost because of their use of the nonconvex loss functions. In addition, compared with SaveBoost, RBoost makes the sizes of such same outliers are smaller due to the designing of numerical stably solver. It still assigns some relatively large weights to more number of points, however, which can certainly affect its performance. As for RobustBoost, the weights of the outliers near to Bayes decision boundary are still larger than that of other points, which means that RobustBoost is somehow affected by outliers. Fourth, based on Fig. 3(g)–(i), one can see that different λ gives different performance in SPLBoost. Specifically, compared with Fig. 3(h), where λ is smaller in Fig. 3(g), the algorithm is more rigorous to the outliers and more samples are considered to be outliers. On the contrary, where λ is larger in Fig. 3(i), the algorithm is more tolerant

TABLE I
CHARACTERISTICS OF THE UCI DATASETS

Data Set	Sample Number	Feature Number	Percentage of Majority Class
adult	48842	14	75.9
bank	45211	17	88.5
blood	748	4	76.2
connect-4	67557	42	75.4
magic	19020	10	64.8
miniboone	130064	50	71.9
ozone	2536	72	97.1
pima	768	8	65.1
parkinsons	195	22	75.4
pb-T-OR-D	102	4	86.3
ringnorm	7400	20	50.5
spambase	4601	57	60.6
titanic	2201	3	67.7
oocMerl4D	1022	41	68.7
st-german-credit	1000	24	70.0
twonorm	7400	20	50.0
vc-2classes	310	6	67.7

to outliers. In addition, it is not hard to see that SPLBoost performs much better than other competing methods.

To summarize, compared with other boosting algorithms, SPLBoost gives more reasonable sample weight distribution and thus can weaken the influence of the outliers in a larger extent.

B. UCI Dataset

In this section, we shall demonstrate the experimental results on 17 UCI datasets. And Table I lists their detailed information.

For every dataset, we randomly select 70% samples as the training data and the rest to be the test data. To evaluate the robustness of those compared boosting algorithms, we randomly select a certain proportion of the training data points and flip their labels. The noise levels are set to be 0%, 5%, 10%, 20%, 30%, respectively. Similar to before, SPLBoost and RobustBoost are chosen as C4.5, while for LogitBoost and RBoost, CART is used as the weak learner. The parameter setting for all the algorithms is the same as in the previous synthetic data experiment. Especially, for SPLBoost, we determine the "age" parameter λ in [1.0, 6.0] with step size 0.1 by using five-fold cross-validation. To illustrate the robust performance of SPLBoost with different SP-regularizers, we implement SPLBoost with four popular SP-regularizers, namely, hard weighting, linear soft weighting, and polynomial soft weighting with $t = 1.3$ and $t = 4.0$, which are denoted as hard, linear, lowerconvex, and upperconvex, respectively. This procedure is repeated for 50 times and the average value of the testing error changing with different noise levels for different boosting algorithms are plotted in Fig. 4.

It can be seen from Fig. 4 that once there have outliers in the training data, the performance of AdaBoost can be heavily depraved and is not comparable with LogitBoost, SavageBoost, RBoost, RobustBoost and SPLBoost, which confirms that AdaBoost is very sensitive to the noisy data. We can also see that for most of the datasets, SPLBoost gives lower test errors than other boosting algorithms, which reveals that SPLBoost has best robustness among all the compared methods. In addition, it is not hard to see that there are no significant difference between the performance of the SPLBoost with four different SP-regularizers.

There are totally 17 UCI datasets and each one has five noise levels, which gives totally 85 cases. For each case, we can rank the performance of those compared algorithms from 1 (the algorithm with the lowest mean testing error) to 9 (the algorithm with the highest mean testing error). Then, we calculate the ratio of the cases in which such a compared algorithm is among the top-n ranked ones and the results are summarized in Fig. 5. For example, the top1 part means for one algorithm there are totally corresponding ratio, namely, (y-axis)$\times$85 cases in which this algorithm can get the best performance, and the top5 part means for one algorithm there are totally corresponding ratio, namely, (y-axis)$\times$85 cases in which this algorithm can get the top5 performance. It can be easily seen that in most cases, SPLBoost performed much better than other competing boosting algorithms, which clearly confirms that it has better resistance to large noise and outliers.

C. Real Application on Webly Labeled Image Data

Using the machine-learning algorithms to classify images usually needs a mass of manually labeled training data. Unfortunately, manually labeling images is a very time-consuming task. To remedy this issue, we can search the categories by using the search engine and get abundant images, then those images corresponding to a specific category can all be labeled by the category and compose the training data. Thus, it is easy for us to get so many various labeled images, which are called webly labeled data. However, training data constructed in this way can be noisy because there may be some images not belonging to the categories we want. As a result, an effective robust classification algorithm is necessary for dealing with the webly labeled training data. In this section, we shall evaluate the proposed SPLBoost algorithm on the webly labeled training data for two image classification tasks.

1) Dogs Versus Cats: Dogs versus Cats[1] [50] is a famous competition released on Kaggle to classify whether images contain either a dog or a cat. This competition contains 12 500 images of dogs and 12 500 images of cats as the training data. In this case, we use the totally 25 000 labeled images to compose the test data, then download some dog and cat images, respectively, using the search engine to compose the training data. As said above, the training data could usually include some outliers, that is, there have some false images in each category. One can see some examples of such outliers in Fig. 6.

The popular googLeNet [51] is used to transform the training and testing images into feature representations. The approaches to be evaluated include AdaBoost, LogitBoost, SavageBoost, RBoost, RobustBoost, SVM, SPBL and SPLBoost. Similar to before, five-fold cross-validation procedure is used to choose the appropriate parameters for all

[1]https://www.kaggle.com/c/dogs-vs-cats

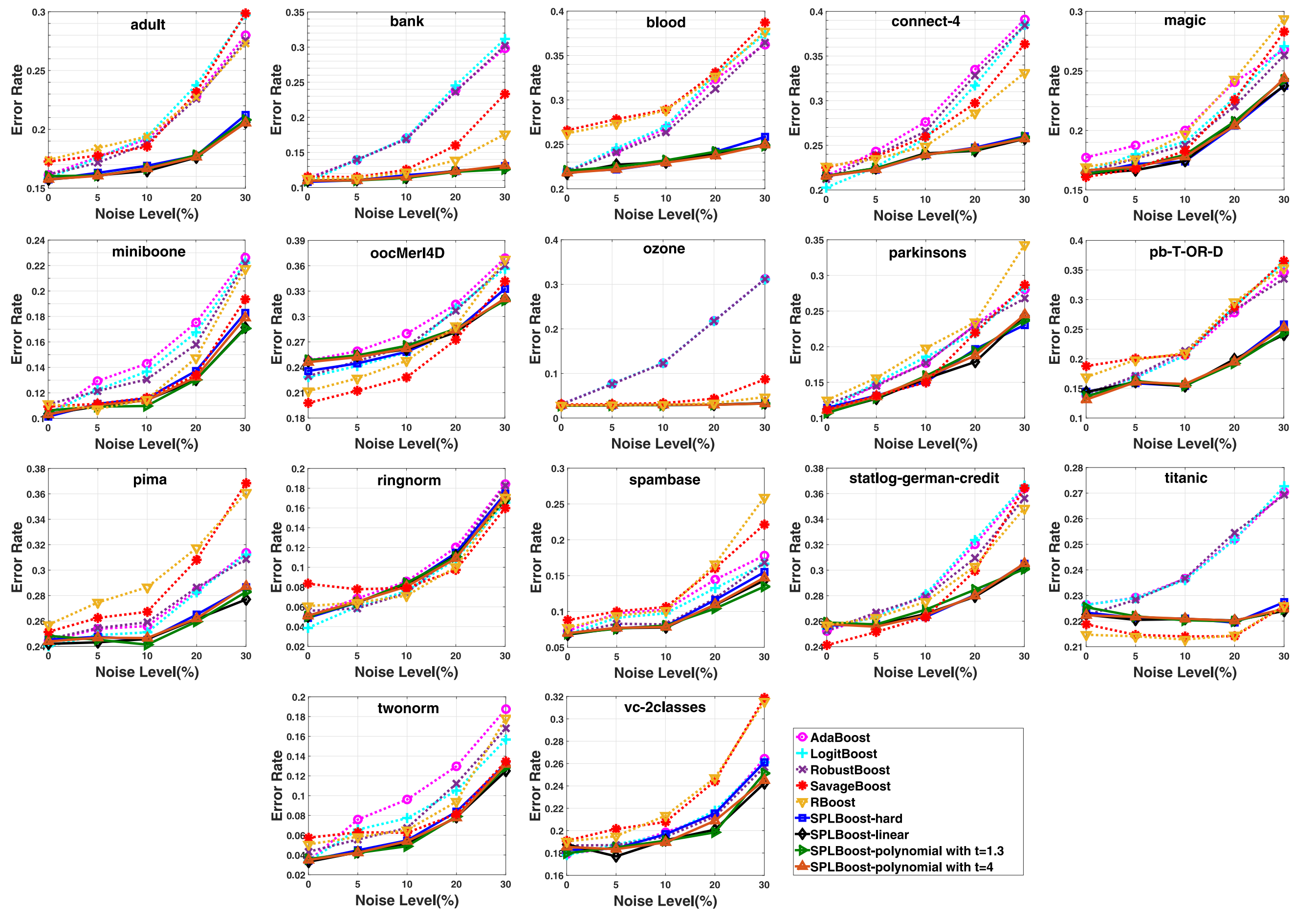


Fig. 4. Testing error rates changing with different noise levels for the various boosting algorithms on 17 UCI datasets.

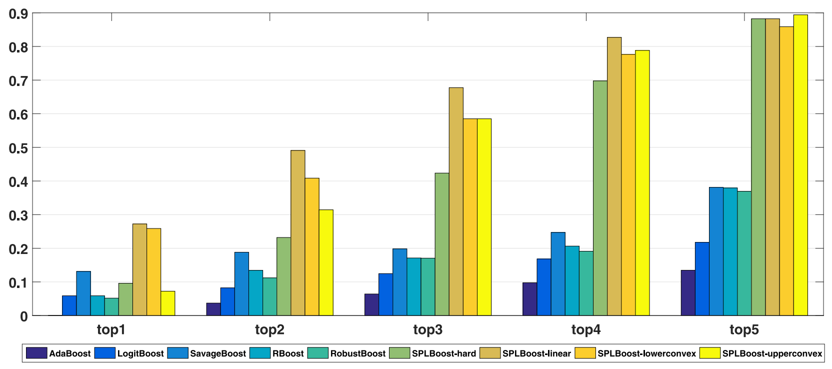


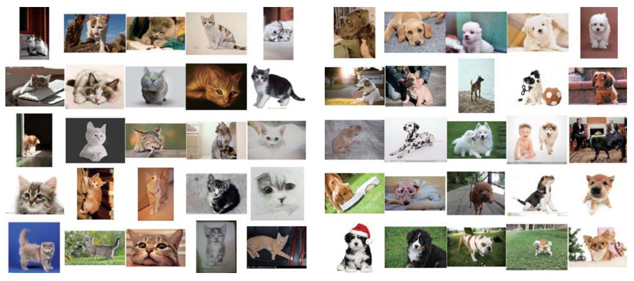
Fig. 5. Statistics of rank values on 17 UCI datasets with five noise levels.

TABLE II
TEST ERROR ON THE DOGS VERSUS CATS DATASET

Algorithm	AdaBoost	LogitBoost	SavageBoost	RBoost	RobustBoost	SVM	SPBL	SPLBoost
Test error	0.0507	0.0450	0.0436	0.0457	0.0364	0.0336	0.0423	**0.0306**

the compared algorithms. And for SPLBoost, the hard weighting is used as the SP-regularizer. The comparison performance in terms of test error is reported in Table II. One can see that SPLBoost gets the lowest test error among all such methods, which supports its superiority in dealing with the real-world dataset with heavy noise/outliers.

As already mentioned, SPLBoost is capable of deleting outliers by assigning zero weights to those doubtful training samples. To verify the identity of the samples deleted by SPLBoost, we display those deleted samples in Fig. 7, here the age parameter λ is set to be 6. It can be easily seen from Fig. 7 that, almost all the samples deleted by SPLBoost in


(a) (b)

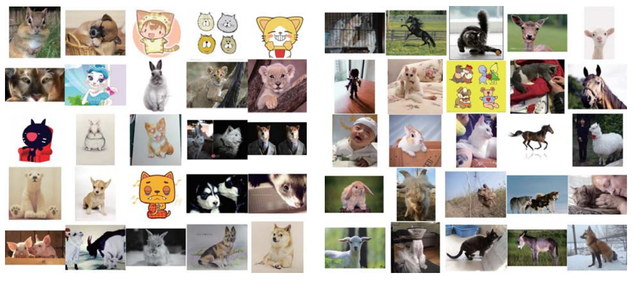

(c) (d)

Fig. 6. Some training samples of Dogs versus Cats data. (a) True images in cat. (b) True images in dog. (c) False images in cat. (d) False images in dog.

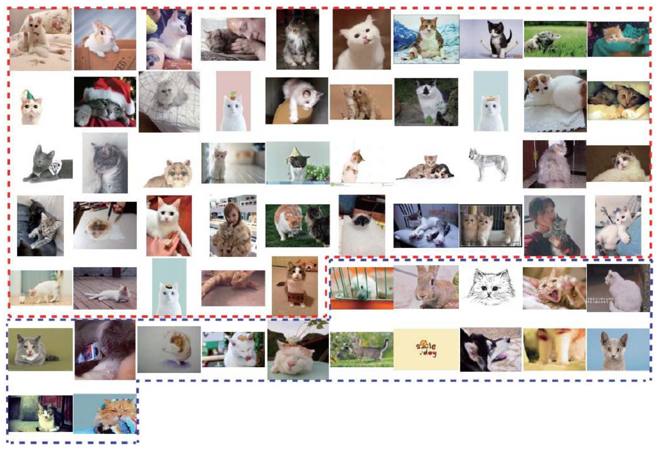


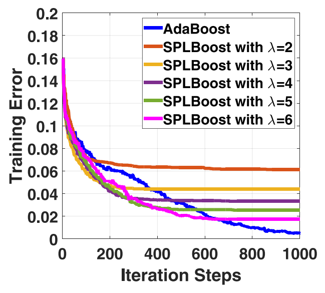
Fig. 7. All the deleted samples in SPLBoost with λ = 6 in Dogs versus Cats experiment, where those images circled by the red dashed line belong to Dog category and those images circled by the blue dashed line belong to Cat category in the training data.

"Dog" category are outliers, that is, those samples are cats, while there some of the deleted samples in "Cat" category are cats. This mainly because that there are less outliers in Cat category than Dog category in this dataset.

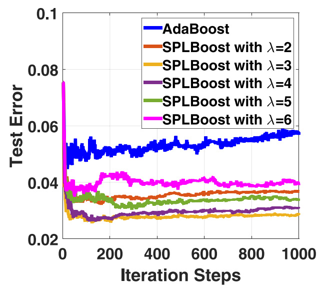
2) Fine-Grained Dog Breed Identification: The Stanford Dogs dataset [52] contains a number of images of 120 breeds of dogs. This dataset has been built using images and annotation from ImageNet [53] for the task of fine-grained image categorization. In this experiment, we select several pairs of breeds which seem similar and are hard to distinguish. Those pairs include beagle versus bloodhound, golden retriever versus labrador, miniature pinscher versus chihuahua, pekinese versus toy poodle, and pug versus french bulldog. In this way, we need to consider five binary classification problems and similar to the previous experiment, for each problem, all the

TABLE III
TEST ERROR ON THE FINE-GRAINED DOG BREED IDENTIFICATION DATA

Algorithm	pair (1)	pair (2)	pair (3)	pair (4)	pair (5)
AdaBoost	0.0550	0.0717	0.1702	0.0300	0.0576
LogitBoost	0.0396	0.0655	0.1691	0.0568	0.0532
SavageBoost	0.0412	0.0636	0.1637	0.0455	0.0490
RBoost	0.0403	0.0605	0.1607	0.0333	0.0476
RobustBoost	0.0339	0.0559	0.1691	0.0300	0.0563
SVM	0.0288	0.0437	0.1369	0.0334	0.0390
SPBL	0.0358	0.0557	0.1428	0.0498	0.0480
SPLBoost	**0.0183**	**0.0314**	**0.1250**	**0.0267**	**0.0362**



(a)



(b)

Fig. 8. (a) Training error and (b) test error along with the iteration steps of AdaBoost and SPLBoost with different age parameter λ.

labeled images corresponding to the dog breed pairs are used to compose the test data, then we search the dog breeds and download the images, respectively, using the search engine to compose the training data. The feature extraction process and experiment settings are the same as in the Dogs versus Cats experiment.

The test errors of all the compared algorithms are listed in Table III, in which pair (1) to pair (5) mean beagle versus bloodhound, golden retriever versus labrador, miniature pinscher versus chihuahua, pekinese versus toy poodle, and pug versus french bulldog, respectively. One can see that SPLBoost can get the lowest test error on the five noisy dataset, which also illustrates its superiority in practice.

D. Discussion

1) Outlier-Resistant of SPLBoost: In the real-world Dogs versus Cats experiment, the labels of test data are actually known and thus we can calculate the training error and test error in every step, which along with the iteration steps of AdaBoost and SPLBoost with different values of λ are plotted in Fig. 8. One can see from Fig. 8(a) that, with the iteration process proceeding, the training error of AdaBoost keeps decreasing, while SPLBoost decreases gradually within a number of iterations and after that it will stop decreasing and keep a constant. This clearly shows that AdaBoost can overfit to the outliers easily, while SPLBoost can effectively reduce the influences of outliers. As we can see from Fig. 8(b), the test error of SPLBoost almost could not increase even in the quite large iteration steps, which proves the robustness of SPLBoost.

2) Convergence Behavior of SPLBoost: In Fig. 9, we plot the quantity of the latent objective function in Theorem 1 with the change of iterations for SPLBoost with different values of

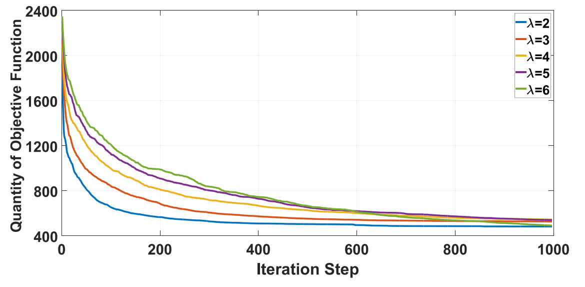


Fig. 9. Quantity of the objective function along with the iteration steps in SPLBoost with the change of λ on Dogs versus Cats data.

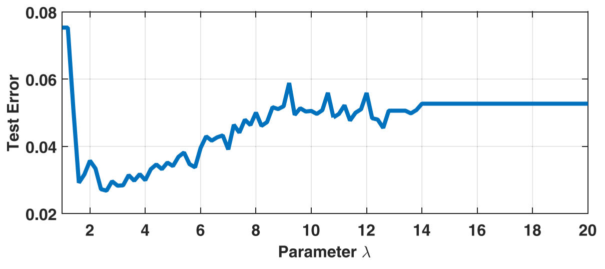


Fig. 10. Test error of SPLBoost with the change of λ on Dogs versus Cats data.

λ on Dogs versus Cats dataset. We can see that for a certain λ, the quantity is always monotonically decreasing during the iteration, which is consistent with the assertion of Theorem 1, that is, a weak convergence result of SPLBoost can be directly obtained.

3) Sensitivity Analysis of Age Parameter λ: To show the sensitiveness of SPLBoost to the age parameter λ, we show the test error of SPLBoost with different λs on Dogs versus Cats dataset in Fig. 10, where λ varies between 1 and 20 with stepsize 0.2, and the test error is the result of the 500th step. One can see that when λ is too small, there have too many samples to be deleted, which gives bad performance. When $\lambda \in [2.0, 6.0]$, the performance is good and stable and after that, the test error increases gradually and finally maintains a constant, where SPLBoost has degenerated into AdaBoost. As a whole, SPLBoost is not so sensitive to the choose of λ.

V. Conclusion

Boosting can be interpreted as an optimization technique to minimize an underlying loss function, and the loss function determines the robustness of the algorithm. The convex loss functions, such as the exponential loss used by AdaBoost and the logistic loss used by LogitBoost, have been proven to be sensitive to the outliers. The nonconvex loss functions, such as Savage loss used by SavageBoost and Savage2 loss used by RBoost, have illustrated superior robustness over popular convex losses, however, solving the nonconvex optimization problem derived from nonconvex losses is not an easy task.

In this article, instead of designing new loss function, we combine the classical discrete AdaBoost algorithm with SPL regime, that is, a robust algorithm framework which has been attracting troumendous attention in machine learning and computer vision. Thus, we come up with a new robust boosting algorithm named SPLBoost. Experiments show that SPLBoost could have a superior performance over other popular ones in dealing with training data with outliers.

However, there are still some interesting works need to be done in the future. On the one hand, it is not hard to see that the SPLBoost can be treated as a general framework to improve the robustness of various boosting algorithms besides AdaBoost. As such, one can try some other popular boosting algorithms, such as LogitBoost, L_2Boost to get better performance. On the other hand, although we have proven the equivalence between the SPLBoost and an MM algorithm implemented on a latent nonconvex objective function, the detailed theoretical properties of SPLBoost, including consistency, convergence rate, and error bound, are needed to be further investigated.

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proc. Eur. Conf. Comput. Learn. Theory*, 1995, pp. 23–37.
[2] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. Cambridge, MA, USA: MIT Press, 2012.
[3] R. Meir and G. Rätsch, "An introduction to boosting and leveraging," in *Advanced Lectures on Machine Learning*. Heidelberg, Germany: Springer, 2003, pp. 118–183.
[4] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Stat.*, vol. 29, no. 5, pp. 1189–1232, 2001.
[5] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. Int. Conf. Mach. Learn.*, vol. 96, 1996, pp. 148–156.
[6] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Mach. Learn.*, vol. 37, no. 3, pp. 297–336, 1999.
[7] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2001, pp. 511–518.
[8] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.
[9] F. Lv and R. Nevatia, "Recognition and segmentation of 3-D human action using HMM and multi-class AdaBoost," in *Proc. Eur. Conf. Comput. Vis.*. 2006, pp. 359–372.
[10] J. C.-W. Chan and D. Paelinckx, "Evaluation of random forest and AdaBoost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery," *Remote Sens. Environ.*, vol. 112, no. 6, pp. 2999–3011, 2008.
[11] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 5, pp. 845–869, May 2014.
[12] T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 3, pp. 552–568, May 2011.
[13] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *J. Artif. Intell. Res.*, vol. 11, no. 1, pp. 131–167, 1999.
[14] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting (with discussion and a rejoinder by the authors)," *Ann. Stat.*, vol. 28, no. 2, pp. 337–407, 2000.
[15] W. Bi, L. Wang, J. T. Kwok, and Z. Tu, "Learning to predict from crowdsourced data," in *Proc. UAI*, 2014, pp. 82–91.
[16] J. Liang, L. Jiang, D. Meng, and A. G. Hauptmann, "Learning to detect concepts from webly-labeled video data," in *Proc. IJCAI*, 2016, pp. 1746–1752.
[17] B. Zhuang, L. Liu, Y. Li, C. Shen, and I. D. Reid, "Attend in groups: A weakly-supervised deep learning framework for learning from Web data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1878–1887.

[18] T. Bylander, "Learning linear threshold functions in the presence of classification noise," in *Proc. ACM 7th Annu. Conf. Comput. Learn. Theory*, 1994, pp. 340–347.

[19] N. D. Lawrence and B. Schölkopf, "Estimating a kernel fisher discriminant in the presence of label noise," in *Proc. ICML*, vol. 1, 2001, pp. 306–313.

[20] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Learning with noisy labels," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1196–1204.

[21] N. Cesa-Bianchi, S. Shalev-Shwartz, and O. Shamir, "Online learning of noisy data," *IEEE Trans. Inf. Theory*, vol. 57, no. 12, pp. 7907–7931, Dec. 2011.

[22] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *arXiv preprint arXiv:1406.2080*, 2014.

[23] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2691–2699.

[24] Y. Wang *et al.*, "Iterative learning with open-set noisy labels," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8688–8696.

[25] C. Domingo and O. Watanabe, "MadaBoost: A modification of AdaBoost," in *Proc. 13th Annu. Conf. Comput. Learn. Theory*, 2000, pp. 180–189.

[26] M. Collins, R. E. Schapire, and Y. Singer, "Logistic regression, AdaBoost and Bregman distances," *Mach. Learn.*, vol. 48, nos. 1–3, pp. 253–285, 2002.

[27] A. H. Li and J. Bradic, "Boosting in the presence of outliers: Adaptive classification with non-convex loss functions," *J. Amer. Stat. Assoc.*, vol. 113, no. 552, pp. 660–674, 2018.

[28] V. Koltchinskii and D. Panchenko, "Empirical margin distributions and bounding the generalization error of combined classifiers," *Ann. Stat.*, vol. 30, no. 1, pp. 1–50, 2002.

[29] T. Zhang and B. Yu, "Boosting with early stopping: Convergence and consistency," *Ann. Stat.*, vol. 33, no. 4, pp. 1538–1579, 2005.

[30] P. M. Long and R. A. Servedio, "Random classification noise defeats all convex potential boosters," *Mach. Learn.*, vol. 78, no. 3, pp. 287–304, 2010.

[31] Y. Freund, "Boosting a weak learning algorithm by majority," *Inf. Comput.*, vol. 121, no. 2, pp. 256–285, 1995.

[32] Y. Freund, "An adaptive version of the boost by majority algorithm," *Mach. Learn.*, vol. 43, no. 3, pp. 293–318, 2001.

[33] Y. Freund, "A more robust boosting algorithm," *arXiv preprint arXiv:0905.2138*, 2009.

[34] H. Masnadi-Shirazi and N. Vasconcelos, "On the design of loss functions for classification: Theory, robustness to outliers, and SavageBoost," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1049–1056.

[35] Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, and J. Song, "RBoost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 11, pp. 2216–2228, Nov. 2016.

[36] T. Pi *et al.*, "Self-paced boost learning for classification," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, New York, NY, USA, Jul. 2016, pp. 1932–1938.

[37] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. ACM 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[38] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 1189–1197.

[39] L. Jiang, D. Meng, T. Mitamura, and A. G. Hauptmann, "Easy samples first: Self-paced reranking for zero-example multimedia search," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 547–556.

[40] Q. Zhao, D. Meng, L. Jiang, Q. Xie, Z. Xu, and A. G. Hauptmann, "Self-paced learning for matrix factorization," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 3196–3202.

[41] H. Li, M. Gong, D. Meng, and Q. Miao, "Multi-objective self-paced learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1802–1808.

[42] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. Hauptmann, "Self-paced learning with diversity," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2078–2086.

[43] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann, "Self-paced curriculum learning," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 2694–2700.

[44] D. Zhang, D. Meng, and J. Han, "Co-saliency detection via a self-paced multiple-instance learning framework," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 5, pp. 865–878, May 2017.

[45] K. D. Tang, V. Ramanathan, F.-F. Li, and D. Koller, "Shifting weights: Adapting object detectors from image to video," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 638–646.

[46] J. S. Supancic and D. Ramanan, "Self-paced learning for long-term tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2379–2386.

[47] L. Lin, K. Wang, D. Meng, W. Zuo, and L. Zhang, "Active self-paced learning for cost-effective and progressive face identification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 1, pp. 7–19, Jan. 2018.

[48] M. P. Kumar, H. Turki, D. Preston, and D. Koller, "Learning specific-class segmentation from diverse data," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 1800–1807.

[49] D. Meng, Q. Zhao, and L. Jiang, "A theoretical understanding of self-paced learning," *Inf. Sci.*, vol. 414, pp. 319–328, Nov. 2017.

[50] J. Elson, J. R. Douceur, J. Howell, J. Saul, and Asirra, "A Captcha that exploits interest-aligned manual image categorization," in *Proc. 14th ACM Conf. Comput. Commun. Security (CCS)*, Oct. 2007, pp. 366–374.

[51] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.

[52] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *Proc. 1st Workshop Fine Grained Vis. Categorization IEEE Conf. Comput. Vis. Pattern Recognit.*, Colorado Springs, CO, USA, Jun. 2011, pp. 1–2.

[53] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, Jun. 2009, pp. 248–255.

Kaidong Wang received the B.Sc. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China, in 2013, where he is currently pursuing the Ph.D. degree with the School of Mathematics and Statistics.

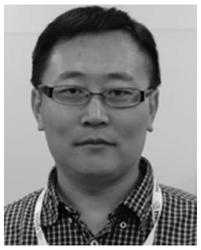
His current research interests include machine learning, deep learning, and hyperspectral image processing.

Yao Wang received the Ph.D. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China, in 2014.

He is currently an Associate Professor with the School of Management, Xi'an Jiaotong University. His current research interests include statistical signal processing, high-dimensional data analysis, and machine learning.

Qian Zhao received the Ph.D. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China, in 2015.

He was a Visiting Scholar with Carnegie Mellon University, Pittsburgh, PA, USA, from 2013 to 2014. He is currently an Associate Professor with the School of Mathematics and Statistics, Xi'an Jiaotong University. His current research interests include low matrix/tensor analysis, Bayesian modeling, and self-paced learning.



Deyu Meng (M'14) received the Ph.D. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, China, in 2008.

He is currently a Professor with the School of Mathematics and Statistics, Xi'an Jiaotong University and an Adjunct Professor with the Faculty of Information Technology, Macau University of Science and Technology, Macau, China. From 2012 to 2014, he took his two-year sabbatical leave with Carnegie Mellon University, Pittsburgh, PA, USA. His current research interests include self-paced learning, noise modeling, and tensor sparsity.

Xiuwu Liao received the Ph.D. degree in management science and engineering from the Dalian University of Technology, Dalian, China, in 2002.

He is currently a Professor with the School of Management, Xi'an Jiaotong University. His current research interests include decision analysis and machine learning, and IT economics.

Zongben Xu received the Ph.D. degree in mathematics from Xi'an Jiaotong University, Xi'an, China, in 1987.

He serves as the Director of the Institute for Information and System Sciences, Xi'an Jiaotong University. His current research interests include intelligent information processing and applied mathematics.

Dr. Xu was elected as a member of Chinese Academy of Science in 2011.