Knowledge-Based Systems 193 (2020) 105424

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Robust boosting via self-sampling☆

Xiaoshuang Liu, Senlin Luo, Limin Pan *

Information System and Security & Countermeasures Experimental Center, Beijing Institute of Technology, Beijing 100081, PR China

ARTICLE INFO

Article history:
Received 13 December 2018
Received in revised form 30 July 2019
Accepted 22 December 2019
Available online 24 December 2019

Keywords:
Boosting
Loss function
Robustness
Self-sampling

ABSTRACT

Boosting is a widely used ensemble meta-algorithm due to its excellent performance in combining weak learners into a strong learner. However, vanilla boosting methods are proved to be sensitive to noise due to the lack of restriction to the always misclassified samples during iterations. In this work, we present a new aspect to overcome the sensitiveness problem by combining the self-sampling learning framework, which can pursue reliable samples and smooth the training process based on the sample reliability measurement designed for the boosting procedure. Experimental results on the synthetic data and several real-world datasets show that the self-sampling regime can automatically optimize an appropriate training subset in different noise environments, and robust boosting algorithms we proposed outperform the state-of-the-art methods.

© 2019 Elsevier B.V. All rights reserved.

1. Introduction

In supervised learning, we look forward to building a stable model that excels in all aspects, but the actual situation is often unsatisfactory, sometimes we can only get multiple models with different strengths. In this case, ensemble learning can provide an effective learning framework for combining multiple models to strong ones. As an important and effective ensemble learning method, boosting technique often makes ensemble classifier performs better than single best classifier [1], and it has been widely applied in images, bioinformatics, text categorization [2–5], and many other fields.

While boosting has evolved somewhat over the years, many different boosting algorithms had been developed [6–8] and among those, AdaBoost still is the most influential one. AdaBoost fits an additive model $F(x) = \sum_{m=1}^{M} c_m f_m(x)$ based on the exponential loss $L(y, F(x)) = \exp(-yF(x))$, where $f_m(x)$ is the weak learner trained on weighted dataset in mth iteration and c_m is a coefficient calculated based on the performance of $f_m(x)$. Because of the loss function, the adjustment of training sample weights also happens at an exponential rate. AdaBoost can generalize well in most cases and it is surprising that even the training loss is reduced to zero, the strong learner can still be improved to give larger classification margins. However, a long-lasting discussion of AdaBoost is its overfitting behavior, which makes prediction rule concentrates too much on the noisy data due to the weight adjustment mechanism with no restrictions. In other words, when the training set contains noise samples, such as in many medical applications [9], it is difficult to classify noise points correctly, which leads to the continuous increase in the weights of noise points until they are overfitted.

To overcome the above disadvantages, we introduce a robust boosting method named Self-sampling Boosting (SSBoost). In doing so, our paper makes the following contributions: (1) A new learning framework named self-sampling learning is proposed, which can adaptively select the reliable and smooth subset of training data in each iteration to learn the weak learner. (2) Based on the self-sampling framework, two robust and well-behaved SSBoost methods are proposed, including the SSAdaBoost and SSGB. (3) The performance of the self-sampling learning and SSBoost algorithms are evaluated experimentally, and the results are compared with other advanced robust boosting algorithms.

The article is organized as follows. Section 2 review several previous studies on robust boosting algorithms and the self-paced learning we are inspired by. In Section 3 we introduce the new self-sampling algorithm together with the SSBoost including SSAdaBoost and SSGB. Then in Section 4 the performance and properties of SSBoost methods are analyzed in empirical evaluation on synthetic and real-world tasks. Finally, we conclude this study in Section 5.

2. Related work

2.1. Robust boosting algorithms

There is a wide variety of methods focused on the classification performance of the AdaBoost on the dataset containing label

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2019.105424.

* Corresponding author.
E-mail addresses: m.liuxsh@gmail.com (X. Liu), luosenlin2012@gmail.com (S. Luo), panlimin2016@gmail.com (L. Pan).

https://doi.org/10.1016/j.knosys.2019.105424
0950-7051/© 2019 Elsevier B.V. All rights reserved.

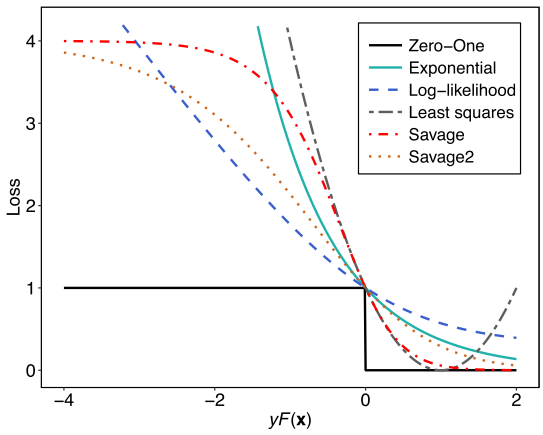


Fig. 1. Loss functions for estimating $F(\boldsymbol{x})$ for classification.

noises. Compared to other ensemble learning methods, boosting primarily reduces bias, and also variance in supervised learning. AdaBoost has been verified to outperform Bagging and Randomization when the training set contains no noise [10], but when the noise is introduced, it performs worse than other methods. Some theoretical aspects of boosting in presence of noisy data [11] show that given a sufficient training time, AdaBoost can do a perfect fit on training data and obtain a generalization loss comparable to the nearest neighbors algorithm, which is suboptimal for noisy data. Researches also prove that for noisy data, regularization methods can make "boosting forever" performs better. In fact, there are three factors affect the robustness of the boosting methods [12]: (1) the loss function; (2) the way to compute the weak learners for boosting iteration; and (3) the regularization. Driven by these factors, many robust boosting algorithms have been proposed.

In [6], the most commonly used version of the AdaBoost [13], which they called Discrete AdaBoost was proved to fit an additive logistic regression model by adaptive Newton updates for minimizing the exponential loss $\mathbb{E}(\exp(-yF(x)))$, and the Real AdaBoost fits an additive logistic regression model by using stagewise optimization of the same expected risk. Compared with the exponential loss, the log-likelihood criteria is monotone and smooth as well and can drive the estimates towards purer solutions even after the training error is zero. Based on that facts, Friedman et al. proposed the LogitBoost algorithm in [6]. The LogitBoost uses an adaptive Newton algorithm to fit additive symmetric logistic regression models under the Bernoulli log-likelihood criteria. As shown in Fig. 1, the log-likelihood loss (suitably scaled) assigns fewer penalties to misclassification samples compared with the exponential loss, which helps the LogitBoost to be more robust than the AdaBoost. Besides that, Friedman et al. also proposed another robust boosting algorithm that uses Newton steps rather than exact optimization for minimizing $\mathbb{E}(\exp(-yF(x)))$, called the Gentle AdaBoost or the GentleBoost [6]. The GentleBoost updates the weights by $\exp(-yf(x))$, which is more stable and pays less attention to the noise than AdaBoost. Empirical evidence shows that the GentleBoost can usually achieve similar effects to the LogitBoost and can perform better when the dataset has reliability problems.

As we can see, although those boosting algorithms referred to above have different loss functions or rely on diverse optimization strategies, their robustness still restricts by the unlimited penalty of the misclassification. The essential reason for this phenomenon is that those loss functions are convex. Minimizing a convex loss function is convenient, but it is not robust enough to noise and as a result, those boosting algorithm still can be sensitive to the noise. As shown in [14], for a wide range of convex loss functions, any such boosting algorithm is highly vulnerable to random classification noise. To overcome that disadvantage, a boosting method called the SavageBoost that optimized the conditional expected risk based on the new robust Savage loss is proposed in [15]. It can be denoted by $\phi_{sa}(yF(x)) = \frac{1}{(1+e^{2yF(x)})^2}$, which is similar to that of least squares loss in the region where $|yF(x)|$ is close to zero but rapidly converge to a constant as $yF(x) \to -\infty$ as shown in Fig. 1 (suitably scaled). The Savage loss is unlike all other previous loss functions and suggests that the boosting algorithm designed with it should be more robust to noise points compared to AdaBoost and LogitBoost. Furthermore, Miao et al. proposed the RBoost1 and RBoost2 algorithm based on the Savage2 loss given by $\phi_{sa2}(yF(x)) = \frac{1}{(1+e^{yF(x)})^2}$ in [12], the Savage2 loss differs from the Savage loss by a second-order factor and it makes the new loss function gentler for misclassification samples and convenient for numeric optimization. Similar to Bartlett and Traskin [16] proved that the AdaBoost is Fisher-consistent, they proved that the Savage2 loss function is consistent as well. That is to say, the minimizer of the conditional expected risk based on these loss functions converges to the Bayes decision rule in the extreme condition, which is the optimal decision for classification problems. Other robust boosting methods based on the Gudermannian loss [17] given by $\phi(yF(x)) = [2\sin^{-1}(\tanh(yF(x))) - 1]^2$ or direct optimize the 0–1 loss [18] also focus on optimizing non-convex loss functions to give a robust and stable classification. Besides, there are also some methods construct an EM algorithm or self-learning to detect noisy instances [19,20] and improve the robustness of the algorithm by optimizing the training data.

Most of the existing robust boosting algorithms we already mentioned try to design various robust loss functions to mitigate or restrict penalties on misclassified samples. However, convex loss functions could not be robust enough for noise, while non-convex loss functions need to be optimized carefully to keep weak learners be numerically stable. Therefore, instead of changing the loss function of AdaBoost, Sabzevari et al. [21] consider using the performance of base learners to correct the label of the noise sample. By inverting the label of the suspect sample, the weight increase of the noise sample can be effectively suppressed at different noise levels, which can be interpreted as a regularization method. Besides, Pi et al. designed a new robust boosting algorithm called SPLBoost [22] by introducing self-paced learning into AdaBoost. Self-paced learning not only enhances the robustness of boosting but also improves the algorithm optimization process. It is the algorithm most similar to our method so far. To give a further explanation, we shall briefly introduce self-paced learning first in the next subsection.

2.2. Self-paced learning

Self-paced learning (SPL) [23] is a recently raised methodology based on curriculum learning (CL) [24] which is inspired by the learning principle of humans and animals. The philosophy under curriculum learning and self-paced learning are to start by learning easier examples of a learning task and then gradually takes more complex samples into the training. Obviously, the key to learning from easy to hard is to find a proper learning order of training samples, so it is important to design a high-quality ranking function for particular problems in the real world. Instead of manually determine the learning order of training samples in curriculum learning, self-paced learning can formulate the ad-hoc idea as a concise model through introducing a self-paced regularizer into the objective function [25]. Such amelioration leads SPL to automatically optimize an appropriate learning process by the

model itself. The formulation of self-paced learning is as follows:

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

where $L(y_i, F(\boldsymbol{x}_i, \boldsymbol{w}))$ stands for the loss function which calculates the cost between the ground truth label y_i and the estimated label $F(\boldsymbol{x}_i, \boldsymbol{w})$, and λ is an "age parameter" for controlling the training scale in every pace while $f(v_i, \lambda)$ denotes the self-paced regularizer whose intrinsic conditions have been theoretically abstracted in [25,26] as follows:

Definition 1 (*Self-paced Regularizer*)**.** Suppose that v is a weight variable, ℓ is the loss, and λ is the age parameter. $f(v;\lambda)$ is called a self-paced regularizer, if

1. $f(v;\lambda)$ is convex with respect to $v\in[0,1]$;
2. $v^*(\ell,\lambda)$ is monotonically decreasing with respect to ℓ, and it holds that $\lim_{\ell\to 0} v^*(\ell,\lambda)=1$, $\lim_{\ell\to\infty} v^*(\ell,\lambda)=0$;
3. $v^*(\ell,\lambda)$ is monotonically increasing with respect to λ, and it holds that $\lim_{\lambda\to\infty} v^*(\ell,\lambda)\le 1$, $\lim_{\lambda\to 0} v^*(\ell,\lambda)=0$;

where $v^*(\ell,\lambda)=\arg\min_{v\in[0,1]} v\ell+f(v;\lambda)$.

The three conditions in Definition 1 provide an axiom for constructing a self-paced regularizer. Condition 2 ensures that the model selects simple samples with small ℓ rather than complex samples, and condition 3 links a larger age parameter to the selection of more complex samples, which means that as the λ grows larger, the model will be more maturely. However, condition 1 further guarantees the convenience of the optimization for the self-paced regularizer. By adopting an alternative optimization strategy (AOS) to iteratively update $\boldsymbol{v}$ and $\boldsymbol{w}$ in (1) with gradually increase the age parameter value, a satisfactory optimization to the objective is expected to be progressively approached. Furthermore, this solving strategy has been proved exactly complies with the known majorization minimization (MM) algorithm [27] on a latent self-paced learning objective in [28].

Under this definition, multiple self-paced regularizers have been constructed. The following lists the hard, linear and mixture self-paced regularizers proposed in [23,25,26] respectively, together with their closed-form solutions $v^*(\lambda,\ell)$ as defined in Definition 1:

$$\begin{aligned}
&f^H(v,\lambda)=-\lambda v;\ \ v^*(\ell,\lambda)=\begin{cases}1, & \text{if } \ell<\lambda\\ 0, & \text{if } \ell\ge\lambda\end{cases}\\
&f^L(v,\lambda)=\lambda(\tfrac{1}{2}v^2-v);\ \ v^*(\ell,\lambda)=\begin{cases}-\ell/\lambda+1, & \text{if } \ell<\lambda\\ 0, & \text{if } \ell\ge\lambda\end{cases}\\
&f^M(v,\lambda,\gamma)=\tfrac{\gamma^2}{v+\gamma/\lambda};\ \ v^*(\ell,\lambda,\gamma)=\begin{cases}1, & \text{if } \ell\le(\frac{\lambda\gamma}{\lambda+\gamma})^2\\ 0, & \text{if } \ell\ge\lambda^2\\ \gamma(\frac{1}{\sqrt{\ell}}-\frac{1}{\lambda}), & \text{otherwise.}\end{cases}
\end{aligned} \tag{2}$$

Inspired by self-paced learning, Pi et al. combined the classical AdaBoost with the hard self-paced regularizer and came up with a new robust boosting method named SPLBoost in [22], but there is an unsatisfactory situation. At the beginning of the boosting, an initialization label or the first weak learner will be assigned or trained. However, due to the limitation of the fitting capability of the base learner, only the most effective but simple rule will be caught at first, which may make the self-paced regularization that only based on the loss be misleading. This phenomenon can be observed more directly in Fig. 2, since that the weak learner in the Discrete AdaBoost is restricted in $\{-1, 1\}$, the losses of the samples can only be e^{-c_1} or e^{c_1} in the first iteration step, where c_1 is the weight of the first weak learner. Considering that $v_i=0$ for the samples whose losses are larger than λ, there would be too many samples could not be selected for the next step if λ falls in $[e^{-c_1}, e^{c_1}]$. At the same time, even in an extremely simplified situation, it is difficult to distinguish simple samples, complex samples or noise only based on the loss. To avoid this kind of unsatisfied situation, one can adopt a warm start procedure, for example, set λ to be a very large number in the first three iterations [22]. Although this trick can be effective in many cases, we prefer to reconsider a more elegant solution. Let us go back to Fig. 2, though the absolute value of loss cannot distinguish samples well, the change of loss may provide additional capacity for pursuing reliable samples and our self-sampling learning is proposed based on this new idea. We will introduce the detail of self-sampling technology and the SSBoost in the next section.

3. Proposed method

3.1. Self-sampling learning

Self-sampling method pays attention to the learning outcome and learning speed of every training sample simultaneously. The learning outcome is represented by the loss as the same in the self-paced learning, while the learning speed will be valued by the change of loss in each iteration. Balancing those two constraints can improve the performance of reliable samples selection whether it is at the beginning or end of the boosting procedure.

Given a set of n binary-class training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, where $\boldsymbol{x}_i\in\mathbb{R}^d$ is the vector-valued features of sample i, $y_i\in\{-1, 1\}$ is the label of $\boldsymbol{x}_i$. Standard-based the supervised learning process, a classification model is trained through the following optimization problem:

$$\min_{\Theta} \mathbf{E}(\Theta)=\sum_{i=1}^{n} L(y_i,F(\boldsymbol{x}_i,\Theta)), \tag{3}$$

where $L(y_i, F(\boldsymbol{x}_i, \Theta))$ denotes the loss function which can be diverse in different boosting algorithms. Considering the boosting as an iterative update process of the classifier, the strong learner $F_M(\boldsymbol{x}_i,\Theta)=\sum_{m=1}^{M} c_m f(\boldsymbol{x}_i,\theta_m)$ is combined by a new base learned in each iteration. For convenience of notions, we briefly write $F_m(\boldsymbol{x}_i,\Theta)$ as F_m in the following. The self-sampling learning focuses on the training speed and training outcomes of every sample during each update which can be formulated as following in the mth iteration:

$$\min_{\Theta,\boldsymbol{v}\in\{0,1\}^n} \mathbf{E}_m(\Theta,\boldsymbol{v};\alpha,\lambda)=\sum_{i=1}^{n} v_i L(y_i,F_m)-\alpha v_i L(y_i,F_{m-1})-\lambda v_i, \tag{4}$$

where the λ is the sampling parameter which determine how many samples can be selected in the next training step and α is a balance parameter to control the strength of two types of constraints.

To solve (4), we note that each variable v_i is allowed to take value in {0,1}, and the loss function such as the exponential loss are convex. In this case, objective (4) is a non-convex optimization problem. However, as in self-paced learning, we still can solve it by using alternative optimization strategy (AOS) [28], which alternatively optimizes the boosting parameters Θ and self-sampling parameters $\boldsymbol{v}$ while keeping the other set of variables fixed. We found in our experiments that AOS obtained accurate results.

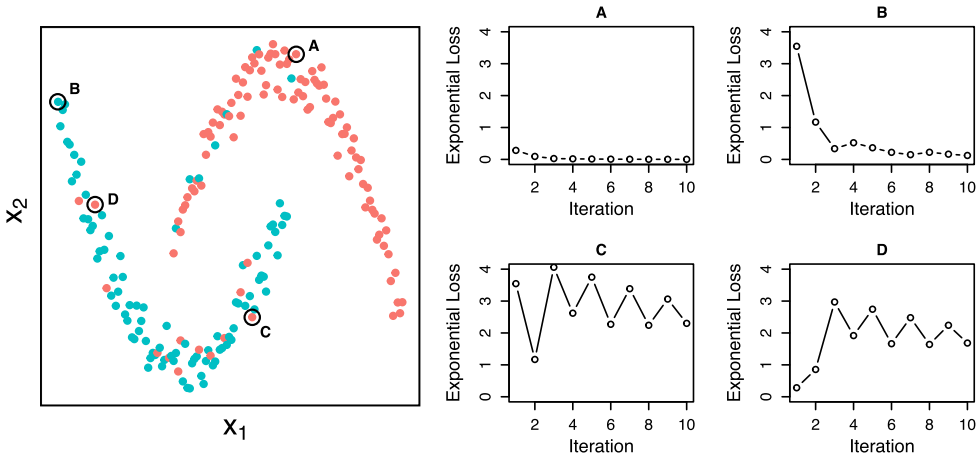


Fig. 2. Different patterns of the loss variation in AdaBoost.

With fixed $\boldsymbol{v}$, problem (4) is a weighted loss minimization problem:

$$\Theta = \arg\min_{\Theta} \sum_{i=1}^{n} v_i L(y_i, F_m(\boldsymbol{x}_i, \Theta)), \tag{5}$$

which is similar to the minimization problem in the traditional boosting algorithm and can be solved by the same optimization process.

With boosting parameters Θ fixed, we can update the self-sampling variable $\boldsymbol{v}$ by solving the problem:

$$\boldsymbol{v} = \arg\min_{\boldsymbol{v}} \sum_{i=1}^{n} v_i L(y_i, F_m) - \sum_{i=1}^{n} (\alpha L(y_i, F_{m-1}) + \lambda) v_i. \tag{6}$$

After fixing boosting parameters, (6) is a convex function of $\boldsymbol{v}$, and thus the global minimum can be obtained at $\nabla_v \mathbf{E}_m(\boldsymbol{v}) = \mathbf{0}$. For convenience of notions, we briefly write $L(y_i, F_m)$ as L_m^i, now we have:

$$\begin{aligned} \frac{\partial \mathbf{E}_m}{\partial v_i} &= L_m^i - \alpha L_{m-1}^i - \lambda \\ &= \alpha(L_m^i - L_{m-1}^i) + (1-\alpha)L_m^i - \lambda \\ &= \alpha \triangle L_m^i + (1-\alpha)L_m^i - \lambda. \end{aligned} \tag{7}$$

The closed-form optimal solution is then given by:

$$v_i^* = \begin{cases} 1, & \alpha \triangle L_m^i + (1-\alpha)L_m^i < \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

From the above derivation, we can see that the self-sampling technology can determine the weight of a sample by measuring a weighted summation of the learning outcome which is represented by L_m^i and the learning speed which is represented by $\triangle L_m^i$. It leads to an update of the original self-paced learning and makes complex but learnable samples have more opportunity to be followed in earlier iterations.

The hyperparameter $\alpha \in [0, 1]$ controls the focus of the self-sampling learning. When α is set to 0, it becomes the same as the hard self-paced learning, while setting α to 1 makes the constraint be only affected by $\triangle L_m^i$. So the appropriate parameter update strategy is to gradually change α from about 0.5 to a smaller value after an appropriate standardization of the $\triangle L_m$ and L_m. In that way, classifiers can focus on heterogeneous patterns in the training data in the early stages of training and avoid too much attention to difficult samples that may be noise points in the later iterations.

However, for the hyperparameter λ, we recommend adjust it by selecting a fixed number or proportion of training samples which is much different from the self-paced learning. This approach we proposed no longer emphasizes the way of learning from a few samples to more samples, two main reasons are as follows. The first reason is that in the boosting algorithm, weak learners need to focus on the high loss samples, so filtering overly complex subsets of possible noise is better than just keeping simple samples. Another reason is to avoid designing the learning sequence. How to determine the size of the training set in each iteration of self-paced learning has always been a difficult problem, and an inappropriate set of learning sequences can have a negative impact on training. Therefore, we consider λ as a sampling rate parameter which determines the proportion of samples kept in the confidence sample selection process.

3.2. SSBoost

Based on the self-sampling learning, we propose two self-sampling boosting algorithms, SSAdaBoost and SSGB.

The original AdaBoost builds an additive logistic regression model based on the exponential loss and seeks a new weak learner $f(\boldsymbol{x})$ in each iteration through the following optimization problem:

$$\{c, f\} = \arg\min_{c,f} \sum_{i=1}^{n} e^{-y_i(F(\boldsymbol{x_i}) + cf(\boldsymbol{x_i}))}. \tag{9}$$

Embedding (9) into a self-sampling model, SSAdaBoost seeks a weak learner $f(\boldsymbol{x})$ and its weight c under the new self-sampling constraint through the following problem:

$$\begin{aligned} \{c, f, \boldsymbol{v}\} = \arg\min_{c,f,\boldsymbol{v}} \sum_{i=1}^{n} & v_i e^{-y_i(F(\boldsymbol{x_i}) + cf(\boldsymbol{x_i}))} \\ & - \alpha v_i e^{-y_i F(\boldsymbol{x_i})} - \lambda v_i, \end{aligned} \tag{10}$$

where α is a balance parameter and λ is a sampling rate parameter. As mentioned in Section 3.1, we can solve (10) by using AOS which alternately updates the learner $F(\boldsymbol{x})$ and the self-sampling weights $\boldsymbol{v}$. We regard the strong learner as the object of optimization, so updating $F(\boldsymbol{x})$ means to train and add a new weighted weak learner $f(\boldsymbol{x})$ to the boosting model. In this way, the optimization process is actually to alternately train a new weak learner $f_m(\boldsymbol{x})$ and update the self-sampling weights, which makes the total number of iterations of SSAdaBoost equal to the original AdaBoost.

In every iteration, we first fixed $\boldsymbol{v}$, which also makes the subpart $(-\alpha v_i e^{-y_i F(\boldsymbol{x_i})} - \lambda v_i)$ of (10) a constant value. Then, (10)

is now a weighted exponential loss minimization problem:

$$\{c, f\} = \arg\min_{c,f} \sum_{i=1}^{n} v_i e^{-y_i(F(\boldsymbol{x_i})+cf(\boldsymbol{x_i}))}. \tag{11}$$

The above (11) is similar to the optimization problem (9) , thus we can solve it by using the similar idea of AdaBoost. Following the way in [6], we build the weighted additive logistic regression model via Newton-like updates for minimize $E(ve^{-yF(\boldsymbol{x})})$. With fixed c (and $\boldsymbol{x}$), we expand (11) to second order about $f(\boldsymbol{x_i}) = 0$:

$$\begin{aligned} L(F + cf) &= \sum_{i=1}^{n} v_i e^{-y_i(F(\boldsymbol{x_i})+cf(\boldsymbol{x_i}))} \\ &\approx \sum_{i=1}^{n} v_i e^{-y_i F(\boldsymbol{x_i})}(1 - y_i cf(\boldsymbol{x_i}) + c^2 y_i^2 f(\boldsymbol{x_i})^2/2) \\ &= \sum_{i=1}^{n} v_i e^{-y_i F(\boldsymbol{x_i})}(1 - y_i cf(\boldsymbol{x_i}) + c^2/2), \end{aligned} \tag{12}$$

since $y^2 = 1$ and $f(\boldsymbol{x})^2 = 1$. Minimizing pointwise with respect to $f(\boldsymbol{x}) \in \{-1, 1\}$ and for $c > 0$, we have

$$\begin{aligned} f &= \arg\min_{f} \sum_{i=1}^{n} v_i e^{-y_i F(\boldsymbol{x_i})}(1 - y_i cf(\boldsymbol{x_i}) + c^2/2) \\ &= \arg\min_{f} \sum_{i=1}^{n} v_i e^{-y_i F(\boldsymbol{x_i})}(y_i - f(\boldsymbol{x_i}))^2 \\ &= \arg\min_{f} \sum_{i=1}^{n} v_i w_i (y_i - f(\boldsymbol{x_i}))^2, \end{aligned} \tag{13}$$

where $w_i = e^{-y_i F(\boldsymbol{x_i})}$ which is the same as the definition in AdaBoost. Supposing $v_i w_i$ is regarded as the sample weight, minimizing the quadratic approximation to the criterion will still lead to a weighted least-squares choice of $f(\boldsymbol{x_i}) \in \{-1, 1\}$. Thus, one can train the weak learner with the sample weights $\boldsymbol{vw}$ to get the additive model f. Next, because of the objective function $L(F+cf)$ with respect to c is convex, we can directly set its derivative to zero to determine c:

$$\begin{aligned} c &= \arg\min_{c} \sum_{i=1}^{n} v_i w_i e^{-y_i cf(\boldsymbol{x_i})} \\ &= \frac{1}{2} \log \frac{1 - \text{err}}{\text{err}}, \end{aligned} \tag{14}$$

where $\text{err} = \sum_{y_i \neq f(\boldsymbol{x_i})} v_i w_i$. Note that c can be negative if the weak learner does worse than 50% under the weighting. Just as we expected, the formula of c in (14) is also consistent with AdaBoost except for the additional self-sampling weight $\boldsymbol{v}$.

After that, with fixed c and f, (10) is a optimization problem about $\boldsymbol{v}$ which can be easily calculated by the following formulation which has been derived in (8):

$$v_i^* = \begin{cases} 1, & \alpha(e^{-y_i(F_i+cf_i)} - e^{-y_i F_i}) + (1-\alpha)e^{-y_i(F_i+cf_i)} < \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

Note that the strong learner $F(\boldsymbol{x})$ has not been updated yet. By simplifying the above formula, we can get a more concise form:

$$v_i^* = \begin{cases} 1, & e^{-y_i F_i}(e^{-y_i cf_i} - \alpha) < \lambda, \\ 0, & \text{otherwise,} \end{cases} \tag{16}$$

where we can observe that the proposed method selects training samples based on the effect of the history learning outcome represented by $e^{-y_i F_i}$ and the latest learning speed which is equivalent to the weighted loss in the new weak learner represented by $e^{-y_i cf_i}$.

Finally, we can get the update of $F(\boldsymbol{x})$ and w_i which is as the same in AdaBoost:

$$F(\boldsymbol{x}) \leftarrow F(\boldsymbol{x}) + cf(\boldsymbol{x}), \tag{17}$$

$$w_i \leftarrow w_i e^{c * 1_{y_i \neq f(\boldsymbol{x_i})}}. \tag{18}$$

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

Algorithm 1 SSAdaBoost algorithm

Input: training samples $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$, iteration count M, sampling proportion μ, balance parameter α, aging parameter δ;

Initialization: $F_0(\boldsymbol{x_i}) = 0$, $w_i = 1/n$, $v_i = 1$;

1: **for** $m = 1$ to M **do**
2: Fit a classifier $f_m(\boldsymbol{x}) \in \{-1, 1\}$ using weights $v_i w_i / \sum_i v_i w_i$ on the training set;
3: Compute $\text{err} = \sum_{y_i \neq f_m(\boldsymbol{x_i})} v_i w_i / \sum_i v_i w_i$, $c_m = \frac{1}{2} \log \frac{1-\text{err}}{\text{err}}$;
4: Calculate λ_m based on μ;
5: Update $v_i = \begin{cases} 1, & e^{-y_i F_{m-1}(\boldsymbol{x_i})}(e^{-y_i c_m f_m(\boldsymbol{x_i})} - \alpha) < \lambda_m, \\ 0, & \text{otherwise;} \end{cases}$
6: Update $w_i \leftarrow w_i e^{c_m 1_{y_i \neq f_m(\boldsymbol{x_i})}}$;
7: Update $F_m(\boldsymbol{x}) \leftarrow F_{m-1}(\boldsymbol{x}) + c_m f_m(\boldsymbol{x})$;
8: Set $\alpha \leftarrow \delta\alpha$;
9: **end for**

Output: the strong classifier sign $F(\boldsymbol{x})$.

Firstly, there is an aging parameter δ in Algorithm 1 which controls the attenuation of the balance parameter α. As mentioned in Section 3.1, α controls the focus of the self-sampling learning, and the appropriate parameter update strategy is to gradually change α from about 0.5 to a smaller value in order to guarantee the self-sampling pays attention to both learning outcomes and learning speed at the beginning of training, while keeping the focus on learning outcomes at the later stage. So in our method, α will be set to 0.5 in the initialization and be updated with $\delta\alpha$ in each iteration, where $\delta \in (0, 1)$. More specifically, an appropriate range of δ is $[0.9, 1)$ and it can be affected by the iteration count M and noise level of the training set, which will be further discussed in our experiments.

Secondly, for choosing the sampling rate parameter λ, it is easy to see that when the adjusted product of the temporary strong learner loss and the new weak learner loss is larger than λ, the self-sampling weight will be zero, which means that sample will not be selected in the next iteration. Thus we call λ the sampling rate who decides how many samples can be selected or trusted in every step. The larger λ is, the more trust the algorithm is in the training set and fewer samples that may be noise will be abandoned. Furthermore, when λ is large enough, the SSAdaBoost degenerates into AdaBoost. As mentioned in Section 3.1, we recommend adjusting λ based on a fixed sampling proportion $\mu \in (0, 1]$ of training samples which means λ needs to be calculated in each iteration. For mth iteration, we first standardize $e^{-\boldsymbol{y}F(\mathbf{X})}$ and $e^{-\boldsymbol{y}c_m f_m(\mathbf{X})}$ after computing c_m, in case α cannot effectively control the balance of the two kinds of constraints. After that, we can get *Lsort* by sorting the value of $e^{-y_i F(\boldsymbol{x_i})}(e^{-y_i c_m f_m(\boldsymbol{x_i})} - \alpha)$ in ascending order and select the μnth value of *Lsort* as the value of λ_m, i.e., λ_m can be represented as

$$\lambda_m = Lsort_{\mu n}. \tag{19}$$

This strategy is similar with the parameter adaptation method in [29]. In addition, another λ adaptation method can start with

ranking $e^{-y_i(F_i+cf_i)} - e^{-y_iF_i}$ and $e^{-y_i(F_i+cf_i)}$ of each sample according to the ascending order separately and then sequence samples incrementally according to the weighted sum of those two rankings based on α and select the μnth sample's value of $e^{-y_iF(\boldsymbol{x_i})}(e^{-y_ic_mf_m(\boldsymbol{x_i})} - \alpha)$ as the value of λ_m. These two methods have shown the similar effect in the experiment.

Finally, comparing with the original AdaBoost, SSAdaBoost modifies the sample weight from $w_i = e^{y_iF(\boldsymbol{x_i})}$ to v_iw_i, which is the reason why SSAdaBoost can be more robust. Weak learners in SSAdaBoost can be trained based on a new stable and low noise dataset benefit from the sample selection of self-sampling learning. Different from the SPLBoost [30], SSAdaBoost can avoid abandoning too many samples due to the low accuracy of the weak learner and does not need a warm start procedure. Based on the above parameter adaptation method, there are only two hyperparameters need to be adjusted besides general boosting hyperparameters, which will basically maintain the computational performance of the boosting algorithm.

Another robust and effective boosting method we proposed is the SSGB. SSGB combines the general Self-sampling learning framework into the gradient boosting which is another important boosting method. Inspired by the idea that boosting can be interpreted as an optimization algorithm on a suitable cost function [31], gradient boosting algorithm is subsequently developed by Friedman et al. [32]. Like other boosting methods, gradient boosting combines weak learners into a strong learner $F(\boldsymbol{x})$ to minimize the object function $\sum_{i=1}^{n} L(y_i, F(\boldsymbol{x_i}))$, however, it applies a steepest descent step to this minimization problem, which leads the model to be updated in accordance with the following equations:

$$\begin{aligned} F_m(\boldsymbol{x}) &= F_{m-1}(\boldsymbol{x}) - c_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(\boldsymbol{x_i})), \\ c_m &= \arg\min_{c} \sum_{i=1}^{n} L(y_i, F_{m-1}(\boldsymbol{x_i}) - c \nabla_{F_{m-1}} L(y_i, F_{m-1}(\boldsymbol{x_i}))), \end{aligned} \tag{20}$$

In each iteration, the weak learner $f_m(\boldsymbol{x})$ is trained to fit the negative gradient of L to update $F(\boldsymbol{x})$, and the coefficient c_m may then be calculated with the aid of line search on the above equations. Although the gradient boosting method already has better robustness than the AdaBoost method, the self-sampling method can further improve the stability of the gradient descent optimization process [33]. The original gradient boosting fits all training samples' gradient to learn a weak learner at every step, which will be affected by the noise points naturally. The self-sampling method can help to draw a relatively "easy" and clean subsample of the training set, and this selected subset is then used, instead of the full sample, to fit the weak learner and compute the model coefficient. In this process, the self-sampling seems like a successive data smooth method which may help to improve the effect of gradient descent intuitively. On the other hand, as shown in Fig. 3, the variation patterns of the sample loss in gradient boosting have shown the same characteristics as the patterns in AdaBoost, where the noise can be detected by the synthesis of training outcomes and speed. So it should be reasonable to combine the gradient boosting with self-sampling constraint for training a robust and efficient strong learner.

The derivation of the SSGB is very similar to that of SSAdaBoost, so we directly give the details of the optimization procedure in Algorithm 2. The self-sampling constraint still has not changed the original structure of gradient boosting, and the parameter adaptation strategies in SSGB are the same as SSAdaBoost.

Algorithm 2 SSGB algorithm

Input: training samples $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, iteration count M, sampling proportion μ, balance parameter α, aging parameter δ;

Initialization: $v_i = 1$, $F_0(\boldsymbol{x_i}) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$, where γ is a constant value;

1: **for** $m = 1$ to M **do**

2: Compute so-called pseudo-residuals:

$$r_{im} = -\frac{\partial L(y_i, F_{m-1}(\boldsymbol{x_i}))}{\partial F_{m-1}(\boldsymbol{x_i})};$$

3: Fit a weak learner $f_m(\boldsymbol{x})$ to pseudo-residuals using weights $\boldsymbol{v}$ on the training set;

4: Compute c_m by solving the optimization problem:

$$c_m = \arg\min_{c} \sum_{i=1}^{n} L(y_i, F_{m-1}(\boldsymbol{x_i}) + cf_m(\boldsymbol{x_i}));$$

5: Calculate λ_m based on μ;

6: Update $v_i = \begin{cases} 1, & \alpha(L_{im} - L_{i(m-1)}) + (1-\alpha)L_{im} < \lambda_m, \\ 0, & \text{otherwise}; \end{cases}$

7: Update $F_m(\boldsymbol{x}) \leftarrow F_{m-1}(\boldsymbol{x}) + c_mf_m(\boldsymbol{x})$;

8: Set $\alpha \leftarrow \delta\alpha$;

9: **end for**

Output: the strong classifier sign $F(\boldsymbol{x})$.

4. Experiments

In this section, we experimentally evaluate the training subset selection ability of the self-sampling learning and the performance of SSBoost algorithms on the synthetic dataset and real-world datasets.

4.1. Synthetic data

For the synthetic dataset, 200 samples for both the positive and negative classes are generated according to two different quadratic functions with the Gaussian noise, and the specific data distribution is the same as shown in Fig. 2. On the synthetic data, two experiments about the effect and characteristics of SSBoost methods are carried out as followed.

In the first experiment, by randomly choosing 30% samples from both classes and then reversing their labels, label noise is added to the original dataset, which has been marked in red in Fig. 4. Then, the strong learners are trained by AdaBoost and SSAdaBoost respectively. For these two algorithms, the weak learner is the Classification And Regression Tree (CART) with a maximum depth of 2. The sampling proportion of μ in SSAdaBoost is set to 0.3, and the aging parameter δ is set to 0.95. We then visualized analyze the sum of every sample's training weight in the first 10 boosting iterations. These experimental results are shown in Fig. 4. The two pictures on the left are the AdaBoost training results and the ones on the right are SSAdaBoost results. In scatter charts, the position of noise points is represented by red points, and the darker the color in the heat map, the larger the accumulation weight of the sample, which means the sample is more concerned by the learner.

By observing Fig. 4, we can see that in the original AdaBoost, it is difficult for weak learners to classify noise points correctly, which leads the weights of noise increase continuously, and the accumulation of weights are much larger than the weights of normal samples. At the same time, because of the influence of noise, weak learners do not have extra energy to focus on the

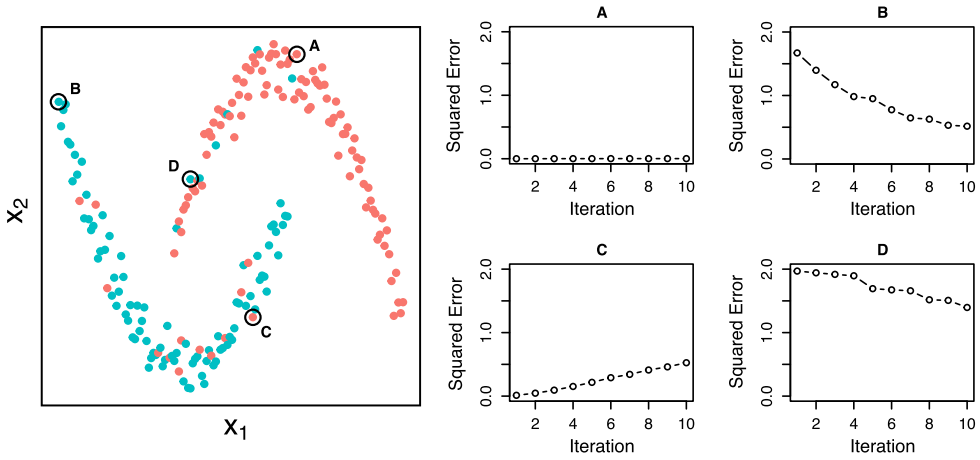


Fig. 3. Different patterns of the loss variation in Gradient Boosting.

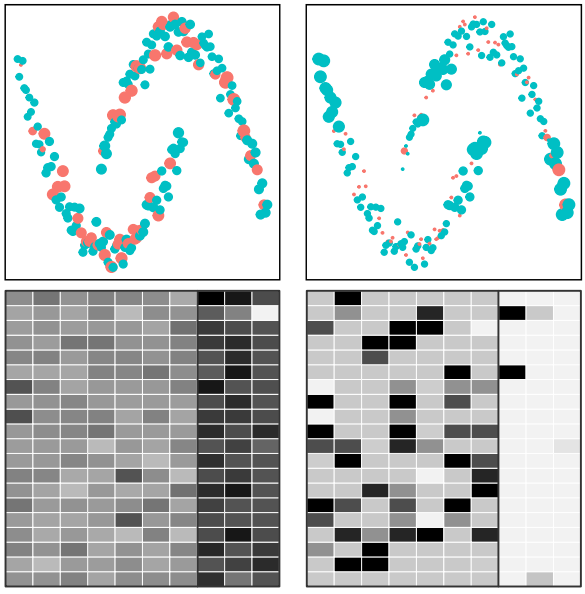


Fig. 4. Accumulation of sample weights in AdaBoost(left) and SSAdaBoost(right). The size of points stands for the accumulation of weights during the iteration and red points represent the noise. Each small square represents a sample, and the three columns on the right side of each heat map show the weights of noise points.

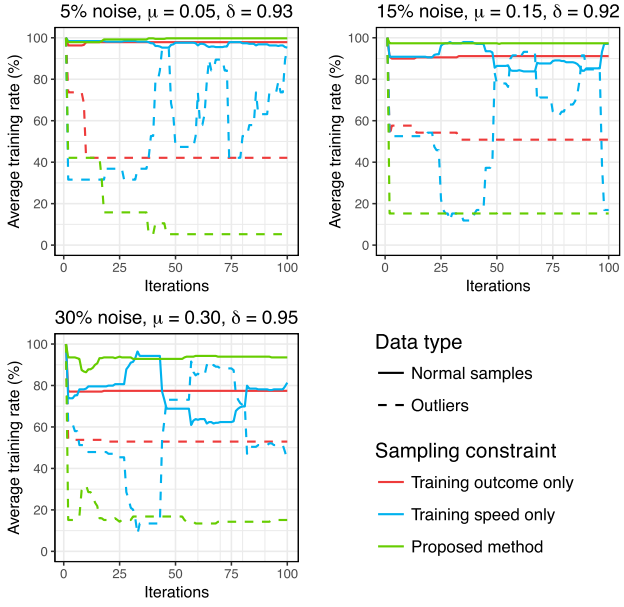


Fig. 5. Self-sampling effect under different constraints.

normal samples in "difficult" locations, which makes the accumulation weights of normal samples are basically the same. On the contrary, constrained by the self-sampling, most of the noise points are not included in training subsets in the SSAdaBoost, so the accumulation weights of noise are far less than those of normal samples. More importantly, the weights of data points on the edge of the function distribution are larger than other samples, which will help the model to learn the accurate discriminant boundary and give larger classification margins.

Next, in the second experiment, we focused on the performance of the SSBoost that utilized both learning outcomes and learning speed at different noise levels. Using the same method as the previous experiment, we add 5%, 15%, and 30% noise to the simulation data and sample the training subsets using three different sampling criteria. One of the sampling criteria is the constraint that only considers learning outcomes similar to the self-paced learning with a fixed age parameter, and it can be achieved by setting the balance parameter $\alpha = 0$ in SSBoost algorithms, which makes the update of $\boldsymbol{v}$ in each iteration only depends on L_m. Another sampling standard only considers the learning speed and may have a better sampling effect for credible samples at the beginning of training. By setting the balance parameter $\alpha = 1$ and aging parameter $\delta = 1$, the update of $\boldsymbol{v}$ will only depend on $\triangle L_m$. The last criterion is our proposed method, which has been described in detail in the previous section. In short, we need to set $\alpha = 0.5$ and $\delta \in [0.9, 1)$ to achieve the self-sampling method. We use the different sampling criteria in the SSGB algorithm, and the weak learner still is the CART with a maximum depth of 2. Like other gradient boosting algorithms, SSGB is regularization by shrinkage which consists in modifying the update rule as $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu \cdot c_m f_m(\boldsymbol{x})$, where parameter $\nu \in (0, 1]$ is called the "learning rate". For each noise level and sampling criterion, SSGB is trained for 100 iterations and the sampling ratio of normal data and noise points is recorded in each iteration. Experiment results are shown in Fig. 5.

From Fig. 5, the following observations can be made. First, our proposed method can effectively identify and filter the noise data under different noise levels. In the three sets of experiments, the sampling method combining learning outcomes and learning speed can make the training subset contain less than 20% of the

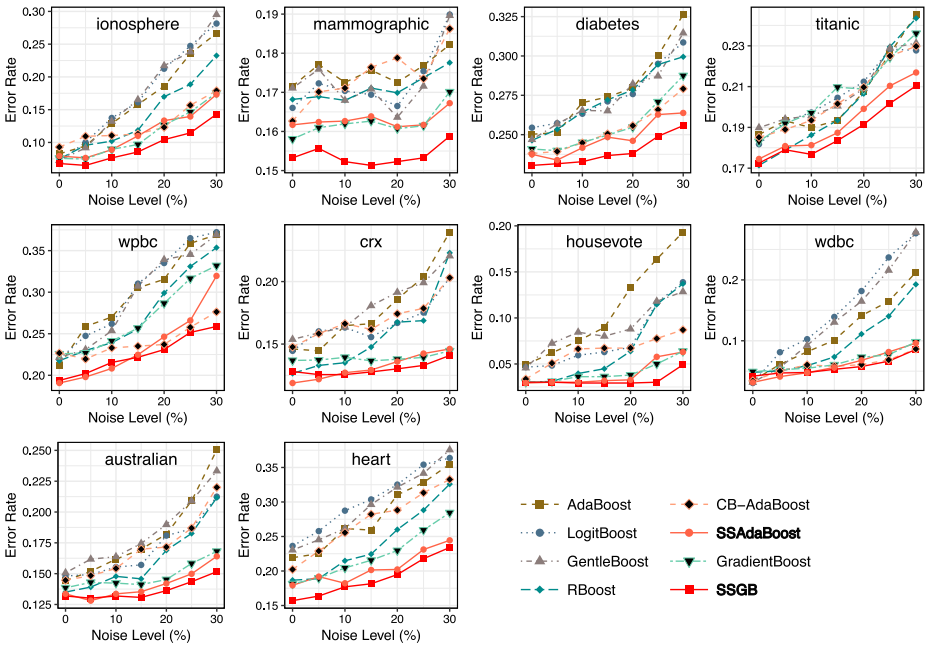


Fig. 6. Testing errors changing with different noise levels for the eight boosting algorithms on ten UCI datasets.

noise data, and ensure that 94%–99% of the normal samples are included in the training. Second, among the three methods, the proposed method has the best stability. Although it seems that the sampling method considering only the learning outcomes can maintain a stable noise filtering ratio throughout the training process, in the repeated experiments, as the random selection of noise data is different, the method only focusing on the outcome will have completely different performance. Therefore, it can be inferred that the sampling method considering only the learning outcome is easy to fall into a locally optimal solution, and the method proposed by us can make use of the instability of the sample learning speed to achieve better sampling results. Third, when we fixed the balance parameters, the aging parameters need to be adjusted to better combine the sampling criteria of learning outcomes and speed and assist the sampling process to jump out of the local optimum. In the later stages of training, the learning speed of the sample will no longer affect the sampling process of the proposed method. Therefore, in the initial stage of training, the effective combination of the two standards is the key to help the sampling get better results, and the aging parameter is the tool to regulate this process.

Experiments on the synthetic dataset prove that the proposed self-sampling method can effectively filter the noise in SSBoost algorithms, and the sampling standard combining the training outcome and the training speed does have better stability and noise filtering performance.

4.2. UCI data

To evaluate the anti-noise ability of SSBoost algorithms quantitatively, we further conduct experiments on ten UCI datasets, including ionosphere, mammographic, diabetes, titanic, wpbc, crx, housevote, wdbc, australian, and heart. In order to compare the robustness and performance of different boosting algorithms, we evaluate the proposed SSBoost algorithms, AdaBoost [6] and some other robust boosting algorithms, including LogitBoost [6], GentleBoost [6], RBoost [12], CB-AdaBoost [34]. At the same time, to compare the effect of SSGB, the comparison algorithm also includes the Gradient Boosting [32].

Our experiment settings are as follows. To evaluate the robustness of those compared boosting methods, we randomly select a certain proportion of the training samples and flip their labels. The noise levels are set to be 0% to 30%. For each noise level, all boosting algorithms are trained for 50 iterations, and 10 times 10*5-fold cross-validation procedure is used to determine the appropriate hyperparameters of some methods and evaluate models performance, which contains an outer loop of 10 folds for error estimation and an inner loop of 5 folds for selection of the best set of hyperparameters.

Specifically, the hyperparameters and their considered values of these methods are as follows. In RBoost, to guarantee the stability of the fitting process, the response z_t is truncated once it exceeds a predefined constant $Z_{max} \in [4, 8]$. Next, in CB-AdaBoost, the k-nearest neighbors(k-NN) is used to compute the agreement rate of labels, and instances with an agreement rate below a certain threshold are eliminated. The threshold is set to 0.21 and the k value in k-NN needs to be adjusted from 2 to 8. Last, the appropriate range of the aging parameter δ in our methods is $[0.9, 1)$, and the sampling proportion $\mu \in [0, 0.4]$ should be adapted to different noise environments (considering that the noise level should be an unknown factor). The weak learner of every boosting algorithm is chosen as a CART with a depth of 2, and the tree completes classification or regression tasks according to specific algorithm requirements. The learning rate or shrinkage of Gradient Boosting and SSGB is set to 0.1. Experiment results are shown in Fig. 6.

As can be seen from Fig. 6, when the weak learner is limited to a tree model with a depth of 2, AdaBoost may still be significantly affected by noise in some datasets, while other robust boosting algorithms can resist the interference of noise points to varying degrees. In the five AdaBoost family comparison methods, RBoost has relatively stable classification performance, and the performance of CB-AdaBoost algorithm is somehow affected by its noise filtering effect in the specific dataset. We can also see that for most of datasets, SSBoost algorithms can give lower test errors than other boosting algorithms including the gradient boosting, which reveals that SSAdaBoost and SSGB have the best robustness among all the compared methods.

To better demonstrate the performance of those compared algorithms, we applied the Nemenyi test [35] for post-hoc test

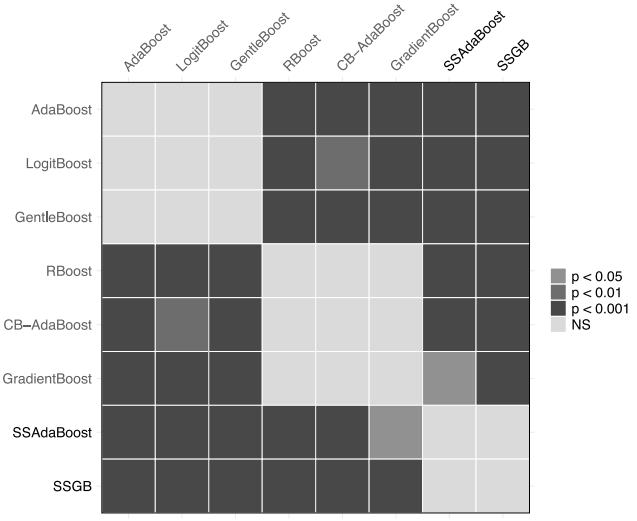


Fig. 7. Comparison of 8 algorithms under the Nemenyi test, NS stands for not significant.

to achieve a comparison of the algorithms. Each comparison algorithm had 70 sets of experimental data (10 UCI datasets multiplied by 7 noise environments), and 8 algorithms are compared to each other under the Nemenyi test. The performance of two classifiers is significantly different if the corresponding average ranked differ by at least the critical difference $CD = q_\alpha\sqrt{\frac{8(8+1)}{6*70}}$, where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. The results are shown in Fig. 7.

As shown in Fig. 7, the existing state-of-the-art techniques including the RBoost and the CB-AdaBoost are significantly better than classic methods and achieve the same level of performance as GradientBoost. At the same time, the SSAdaBoost and SSGB we proposed are further significantly better than the state-of-the-art methods. Although SSAdaBoost and SSGB did not show significant differences, we can see from the p-value that the SSGB has a more significant improvement than previous methods.

4.3. Real-world data

To further verify the effectiveness of proposed algorithms, we collected three larger real datasets covering the areas of healthcare, information security, and climate prediction. The first task is to determine the blood glucose status of non-clinical patients based on the China Health and Nutrition Survey (CHNS).[1] The labels are determined by the fasting blood glucose status of subjects which may be affected by various factors such as fasting time, activity behavior and sample freezing time. The second one is the Android malware recognition task. We collected a large number of malware samples from the public dataset Android Malware Dataset [36] (AMD)[2] and software from the application market as benign samples. We cannot guarantee that the software collected from the market is 100% non-harmful, so if the algorithm can identify these samples, it will help to improve the accuracy of the model. Besides, we use the feature representation method proposed by Zhang et al. [37] to characterize the dataset. The last one is to predict whether some areas of Australia will rain in the coming day.[3] The variability and complexity of the climate will intuitively test the robustness of the learning algorithm. The information of these datasets is shown in Table 1.

[1] https://www.cpc.unc.edu/projects/china.

[2] http://amd.arguslab.org.

[3] https://www.kaggle.com/jsphyg/weather-dataset-rattle-package.

Table 1
Statistical information of the experimental datasets.

Dataset	Sample number	Feature number	Percentage of majority class
CHNS	7931	56	91.6%
AMD	40533	5550	50.8%
Rain	56420	22	78.0%

Table 2
The average classification rate and F1 score of each approach on the three datasets.

	CHNS		AMD		Rain	
	ACC	F1	ACC	F1	ACC	F1
AdaBoost	0.9444	0.6163	0.9100	0.9140	0.8470	0.6140
LogitBoost	0.9470	0.6358	0.8559	0.8593	0.8394	0.5465
GentleBoost	0.9470	0.6358	0.8559	0.8593	0.8419	0.5808
Rboost	0.9520	0.6503	0.9123	0.9145	0.8439	0.5670
CB-AdaBoost	0.9520	0.6460	0.9092	0.9122	0.8439	0.5670
GradientBoost	0.9529	0.6410	0.8924	0.8931	0.8407	0.5175
SSAdaBoost	0.9545	0.6786	**0.9334**	**0.9362**	**0.8477**	**0.6286**
SSGB	**0.9562**	**0.6790**	0.9012	0.9026	0.8423	0.5601

In the experiment, 60% of each dataset is used for training, 20% for verification, and the remaining 20% for testing. The number of iterations of every algorithm is adjusted, ranging from 50 to 200, and the setting for the weak learner and other hyperparameters are the same as in Section 4.2. The experimental results are shown in Table 2.

As we can see from the results, the two algorithms we proposed still perform well in real tasks. However, the experimental results also show some other notable phenomena. Since we did not add extra label noise to these datasets, noise immunity may not be the most important factor for algorithms to perform well. In other words, while the algorithm is robust, it also needs to learn a larger classification margin as much as possible. Therefore, AdaBoost using exponential loss can still perform well on low-noise large-scale data, while the performance of methods using a more robust loss function is not satisfactory. The self-sampling learning method we proposed can enhance the robustness of the algorithm while maintaining the advantage of exponential loss, so the algorithms can still achieve good results.

5. Conclusions

In this paper, we first propose a new reliable sample selection framework called self-sampling learning. This method takes the learning outcome and learning speed of sample as constraints to select a subset from all training data for learning in each iteration, which can remove the noise points accurately and smooth the training process of the model. Second, we design two noise-robust boosting algorithms called SSAdaBoost and SSGB by applying the self-sampling framework to the AdaBoost and GradientBoost, which does not change the structure of the original algorithms but improves the robustness and performance of boosting algorithms effectively. Experimental evaluations on synthetic and real-world datasets show that the self-sampling method that balances learning outcomes and learning speed constraints effectively filters out noise points, and SSBoost algorithms provide better prediction accuracies at multiple noise levels. It is worth mentioning that even in the original dataset without adding noise, the self-sampling can still improve the performance of boosting methods in most cases.

However, there are still some works can be done in future research. First, we will focus on the theoretical properties of the self-sampling learning to figure out whether the loss function contained in this self-sampling constraint is the same as

the self-paced learning that has a close relationship with non-convex regularized penalty (NCRP). Second, we would like to see if the AOS algorithm utilized to solve the self-sampling problem is identical to a majorization-minimization (MM) algorithm implemented on a latent objective function. Besides, although the self-sampling method is designed for the change patterns of sample loss in the boosting process, the framework still has the opportunity to be applied to other learning algorithms, including deep learning. In the future, we will continue to explore the characteristics and applications of the self-sampling method.

Acknowledgments

This work was supported in part by the 13th 5-year National Science and Technology Supporting Project, PR China under Grant 2018YFC2000302 and the National 242 Program, PR China under Grant 2017A149.

References

[1] R. Saini, S.K. Ghosh, Ensemble classifiers in remote sensing: A review, in: 2017 International Conference on Computing, Communication and Automation, ICCCA, 2017, pp. 1148–1152.
[2] D.R. Nayak, R. Dash, B. Majhi, Brain mr image classification using two-dimensional discrete wavelet transform and adaboost with random forests, Neurocomputing 177 (2016) 188–197.
[3] J. Li, P. Luo, G. Zhou, F. Lin, C. Niu, Adaboost with auto-evaluation for conversational models, in: IJCAI International Joint Conference on Artificial Intelligence, vol. 2018-July, 2018, pp. 4173–4179.
[4] C. Zhang, C. Liu, X. Zhang, G. Almpanidis, An up-to-date comparison of state-of-the-art classification algorithms, Expert Syst. Appl. 82 (2017) 128–150.
[5] J. Sun, J. Lang, H. Fujita, H. Li, Imbalanced enterprise credit evaluation with DTE-SBD: decision tree ensemble based on SMOTE and bagging with differentiated sampling rates, Inform. Sci. 425 (2018) 76–91.
[6] J. Friedman, T. Hastie, R. Tibshirani, et al., Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors), Ann. Statist. 28 (2) (2000) 337–407.
[7] G. Biau, B. Cadre, L. Rouvière, Accelerated gradient boosting, Mach. Learn. 108 (6) (2019) 971–992.
[8] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: an open-source software for multi-class imbalance learning, Knowl.-Based Syst. 174 (2019) 137–143.
[9] H. Fujita, D. Cimr, Computer aided detection for fibrillations and flutters using deep convolutional neural network, Inform. Sci. 486 (2019) 231–239.
[10] R.A. McDonald, D.J. Hand, I.A. Eckley, An empirical comparison of three boosting algorithms on real data sets with artificial class noise, in: International Workshop on Multiple Classifier Systems, Springer, 2003, pp. 35–44.
[11] W. Jiang, Some theoretical aspects of boosting in the presence of noisy data, in: Proceedings of the Eighteenth International Conference on Machine Learning, Citeseer, 2001.
[12] Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, 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. 27 (11) (2016) 2216–2228.
[13] Y. Freund, R.E. Schapire, et al., Experiments with a new boosting algorithm, in: Icml, Citeseer, 1996, pp. 148–156.
[14] P.M. Long, R.A. Servedio, Random classification noise defeats all convex potential boosters, Mach. Learn. 78 (3) (2010) 287–304.
[15] H. Masnadi-Shirazi, N. Vasconcelos, On the design of loss functions for classification: theory, robustness to outliers, and savageboost, in: Advances in Neural Information Processing Systems, 2009, pp. 1049–1056.
[16] P.L. Bartlett, M. Traskin, Adaboost is consistent, J. Mach. Learn. Res. 8 (Oct) (2007) 2347–2368.
[17] O. Toka, M. Cetin, A binary boosting method in the presence of outliers in training data, Soft Comput. (2018) 1–10.
[18] T. Nguyen, S. Sanner, Algorithms for direct 0–1 loss optimization in binary classification, in: International Conference on Machine Learning, 2013, pp. 1085–1093.
[19] J. Cao, S. Kwong, R. Wang, A noise-detection based adaboost algorithm for mislabeled data, Pattern Recognit. 45 (12) (2012) 4451–4465.
[20] V.C. Tran, N.T. Nguyen, H. Fujita, D.T. Hoang, D. Hwang, A combination of active learning and self-learning for named entity recognition on twitter using conditional random fields, Knowl.-Based Syst. 132 (2017) 179–187.
[21] M. Sabzevari, G. Martínez-Muñoz, A. Suárez, Vote-boosting ensembles, Pattern Recognit. 83 (2018) 119–133.
[22] T. Pi, X. Li, Z. Zhang, D. Meng, F. Wu, J. Xiao, Y. Zhuang, Self-paced boost learning for classification., in: IJCAI, 2016, pp. 1932–1938.
[23] M.P. Kumar, B. Packer, D. Koller, Self-paced learning for latent variable models, in: Advances in Neural Information Processing Systems, 2010, pp. 1189–1197.
[24] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 41–48.
[25] L. Jiang, D. Meng, T. Mitamura, A.G. Hauptmann, Easy samples first: self-paced reranking for zero-example multimedia search, in: Proceedings of the 22nd ACM International Conference on Multimedia, ACM, 2014, pp. 547–556.
[26] Q. Zhao, D. Meng, L. Jiang, Q. Xie, Z. Xu, A.G. Hauptmann, Self-paced learning for matrix factorization, in: AAAI, 2015, pp. 3196–3202.
[27] K. Lange, D.R. Hunter, I. Yang, Optimization transfer using surrogate objective functions, J. Comput. Graph. Statist. 9 (1) (2000) 1–20.
[28] D. Meng, Q. Zhao, L. Jiang, What objective does self-paced learning indeed optimize? 2015, arXiv preprint arXiv:1511.06049.
[29] H. Li, M. Gong, Self-paced convolutional neural networks, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2017, pp. 2110–2116.
[30] K. Wang, Y. Wang, Q. Zhao, D. Meng, Z. Xu, SPLBoost: an improved robust boosting algorithm based on self-paced learning, 2017, ArXiv abs/1706.06341.
[31] L. Breiman, Arcing the edge, Technical Report 486, Statistics Department, University of California at Berkeley, 1997.
[32] J.H. Friedman, Stochastic gradient boosting, Comput. Statist. Data Anal. 38 (4) (2002) 367–378.
[33] G. Einziger, M. Goldstein, Y. Sa'ar, I. Segall, Verifying robustness of gradient boosted models, in: AAAI, 2019, pp. 2446–2453.
[34] Z. Xiao, Z. Luo, B. Zhong, X. Dang, Robust and efficient boosting method using the conditional risk, IEEE Trans. Neural Netw. Learn. Syst. 29 (7) (2018) 3069–3083.
[35] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[36] F. Wei, Y. Li, S. Roy, X. Ou, W. Zhou, Deep ground truth analysis of current android malware, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA'17, Springer, Bonn, Germany, 2017, pp. 252–276.
[37] H. Zhang, S. Luo, Y. Zhang, L. Pan, An efficient android malware detection system based on method-level behavioral semantic analysis, IEEE Access (2019).