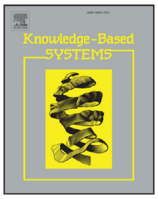
Contents lists available at ScienceDirect

Knowledge-Based Systems

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



A robust multi-class AdaBoost algorithm for mislabeled noisy data

Bo Sun*, Songcan Chen, Jiandong Wang, Haiyan Chen

College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 29 Yudao Street, Nanjing 210016, China

A R T I C L E   I N F O

Article history:
Received 12 June 2015
Revised 23 March 2016
Accepted 25 March 2016
Available online 30 March 2016

Keywords:
Ensemble learning
AdaBoost
Robustness
Multi-class classification
Mislabeled noise

A B S T R A C T

AdaBoost has been theoretically and empirically proved to be a very successful ensemble learning algorithm, which iteratively generates a set of diverse weak learners and combines their outputs using the weighted majority voting rule as the final decision. However, in some cases, AdaBoost leads to overfitting especially for mislabeled noisy training examples, resulting in both its degraded generalization performance and non-robustness. Recently, a representative approach named noise-detection based AdaBoost (ND_AdaBoost) has been proposed to improve the robustness of AdaBoost in the two-class classification scenario, however, in the multi-class scenario, this approach can hardly achieve satisfactory performance due to the following three reasons. (1) If we decompose a multi-class classification problem using such strategies as one-versus-all or one-versus-one, the obtained two-class problems usually have imbalanced training sets, which negatively influences the performance of ND_AdaBoost. (2) If we directly apply ND_AdaBoost to the multi-class classification scenario, its two-class loss function is no longer applicable and its accuracy requirement for the (weak) base classifiers, i.e., greater than 0.5, is too strong to be almost satisfied. (3) ND_AdaBoost still has the tendency of overfitting as it increases the weights of correctly classified noisy examples, which could make it focus on learning these noisy examples in the subsequent iterations. To solve the dilemma, in this paper, we propose a robust multi-class AdaBoost algorithm (Rob_MulAda) whose key ingredients consist in a noise-detection based multi-class loss function and a new weight updating scheme. Experimental study indicates that our newly-proposed weight updating scheme is indeed more robust to mislabeled noises than that of ND_AdaBoost in both two-class and multi-class scenarios. In addition, through the comparison experiments, we also verify the effectiveness of Rob_MulAda and provide a suggestion in choosing the most appropriate noise-alleviating approach according to the concrete noise level in practical applications.

1. Introduction

How to effectively boost the generalization performance of a learner has been an important research topic in machine learning community. Ensemble learning [1–5] is a powerful paradigm for achieving such a goal and has been widely studied and successfully applied in many areas since its invention, e.g., it has been applied in face recognition[6], class imbalance learning [7], concept drift handling [8,9], etc. Many successful ensemble learning algorithms have constantly been proposed in the literature [4,10–17], where Bagging [5,10,12] and AdaBoost [13–15] are most famous. Bagging is a parallel ensemble learning algorithm, which first samples the original training example set $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ ($x_i \in \mathbb{R}^d$, $y_i \in Y$, $d > 1$, Y is the class label set) with replacement to construct multiple diverse bootstrap training subsets $D_{tr\_1}, D_{tr\_2}, \ldots, D_{tr\_T}$, then trains one classifier C_t on each sampled subset D_{tr_t} ($t = 1, 2, \ldots, T$), finally combines these generated classifiers using the majority voting rule to get the final decision. Different from Bagging, AdaBoost is a sequential ensemble learning algorithm and constructs a final ensemble in the following manner: initially, it trains the first classifier C_1 on the original training set D_{tr} whose components have a uniform weight distribution, i.e., $w_i = 1/N$ ($i = 1, 2, \ldots, N$), then the training examples misclassified by classifier C_1 are assigned larger weights, leading to a greater possibility of being selected as components in the training set of the next classifier. That is, the training set of the $(t+1)$th ($1 \le t \le T-1$, T is the ensemble size) classifier g_{t+1} is determined by the classification results of the tth classifier g_t. In this way, AdaBoost focuses more on learning the 'hard' (misclassified) examples and trains classifiers in an iterative way, finally combines the sequentially-generated classifiers using the weighted majority voting rule to obtain the final decision.

In many practical applications, however, the collected training set D_{tr} often contains some 'noisy' examples, e.g., examples

* Corresponding authors. Tel.: +86 158 505 582 15; fax: +86 025 84892956.
E-mail addresses: sunbo87@126.com (B. Sun), s.chen@nuaa.edu.cn (S. Chen), aics@nuaa.edu.cn (J. Wang), chenhaiyan@nuaa.edu.cn (H. Chen).

http://dx.doi.org/10.1016/j.knosys.2016.03.024
0950-7051/Crown Copyright © 2016 Published by Elsevier B.V. All rights reserved.

with class label mislabeled are called *label noises*, or examples with incorrect feature values are called *feature noises*. It has been shown that label noises are generally more detrimental than feature noises to the generalization performance of learned classifiers [18,19]. Thus, in this paper, we mainly investigate the label noises and the term 'noise' that appears in the remaining of this paper refers to the mislabeled noisy training examples. Existing studies have demonstrated that Bagging is insensitive and robust to mislabeled noisy examples while AdaBoost is very sensitive and non-robust to such noises [5,12]. The reason of the poor performance of AdaBoost in the noisy training scenario is probably due to that, compared with the 'clean' (correctly labeled) training examples, the mislabeled noisy ones are more easily misclassified by the current classifier and thus improperly assigned larger weights by the weight updating scheme of AdaBoost, which would make AdaBoost focus more on learning these mislabeled noises and thus overfit them in the subsequent iterations. Therefore, in the mislabeled noisy training scenario, how to effectively improve the robustness of AdaBoost is an important research issue.

Since AdaBoost is originally designed as a two-class classification algorithm, naturally most existing works focus on improving its robustness in the two-class classification problems. A recent work [26] proposed a representative approach named noise-detection based AdaBoost (ND_AdaBoost) to improve the robustness of AdaBoost. However, ND_AdaBoost is hard to be applied in the multi-class classification scenario due to the following three reasons. (1) If we decompose a multi-class classification problem into multiple two-class problems using such strategies as one-versus-all [19] or one-versus-one [20] and then directly apply ND_AdaBoost, the obtained two-class problems usually have imbalanced training sets, which would negatively influence the generalization performance of ND_AdaBoost. (2) If we directly apply ND_AdaBoost to the multi-class classification problems, its two-class loss function is no longer applicable and its accuracy requirement for the base classifiers, i.e., greater than 0.5, is usually too strong to be met. (3) ND_AdaBoost still has the tendency of overfitting as it increases the weights of correctly classified noisy examples in each iteration, which could make it focus on learning these noisy examples in the subsequent iterations.

To solve the above dilemma, we propose a robust multi-class AdaBoost algorithm (Rob_MulAda) based on the popular multi-class AdaBoost algorithm SAMME [21]. In Rob_MulAda, (1) we formally design a noise-detection based multi-class loss function and solve its minimization problem by proving a proposition such that the optimal weight α_T of base classifier g_T ($T \geq 1$) is obtained; (2) we present a new weight updating scheme to alleviate the harmful effect of noisy examples. In the experimental study, we first directly compare ND_AdaBoost with its modified version ND_AdaBoost* that is specifically designed by just substituting the weight updating scheme of ND_AdaBoost with ours to verify the robustness of our weight updating scheme in the two-class classification scenario; Then, we verify the effectiveness of the proposed Rob_MulAda by empirically comparing it with several closely-related algorithms in the multi-class scenario. Experimental results demonstrate that our weight updating scheme is indeed more robust to mislabeled noises than the one used in ND_AdaBoost in both two-class and multi-class classification scenarios, and besides, the proposed algorithm Rob_MulAda is most effective when the mislabeled noise level is low and moderate (no more than 20%). Based on the experimental results, we also provide a suggestion in choosing the most appropriate noise-alleviating approach according to the concrete noise level in practical applications.

The rest of this paper is organized as follows. In Section 2, we briefly review the approaches that have been proposed to enhance AdaBoost's robustness in the two-class classification scenario. In Section 3, we introduce the noise-detection based AdaBoost algorithm ND_AdaBoost for two-class classification problems and Zhu et al.'s multi-class AdaBoost algorithm SAMME to facilitate the later descriptions. In Section 4, we illustrate our robust multi-class AdaBoost algorithm Rob_MulAda in detail. In Section 5 we first empirically verify the robustness of our weight updating scheme in two-class classification problems, and then verify the effectiveness of the proposed Rob_MulAda in multi-class problems. Conclusion and future work are given in Section 6.

2. Related work

Many works have been conducted in the literature to improve the robustness of the classical two-class AdaBoost algorithm in the mislabeled noisy training scenario.

(1) Identifying and removing the noisy training examples is the simplest category of approaches, which first detects the noisy examples in the original training set D_{tr} using a certain method, and then directly removes the detected noisy examples to obtain a refined training set D'_{tr}, after that, AdaBoost algorithm is directly applied on the refined training set. Different methods of detecting noisy examples differentiate concrete approaches in this category.

Muhlenbach et al., [18] proposed an approach that first detects noisy examples based on the class label comparison between each example (x_i, y_i) $(i = 1, 2, \ldots, N)$ and its k (e.g., $k = 5$) nearest neighbors $\{(x_{ij}, y_{ij})\}_{j=1}^{k}$ in the training set, and then removes all the detected noisy examples. Concretely, example (x_i, y_i) is considered to be a noise if its class label y_i is inconsistent with the most prevalent label y among its k nearest neighbors. Their experiments demonstrated that, when introduced 0 to 20% of artificial mislabeled noisy examples into the training set D_{tr}, the approach can improve the generalization performance of learned classifiers to some extent and yields better performance than the approach of relabeling the detected noises. However, this approach shrinks the size of training set D_{tr} and in some cases degrades the generalization performance of the learned classifiers especially when the original size of D_{tr} is small. Angelova et al., [22] proposed an approach which first trains multiple diverse classifiers $\{C_t\}_{t=1}^{T}$ on the original training set D_{tr} using a technique like bootstrap sampling, then detects noisy examples according to the decisions of all the generated classifiers. Concretely, if example (x_i, y_i) $(1 \leq i \leq N)$ causes a large disagreement among the decisions of these classifiers: $C_1(x_i), \ldots, C_T(x_i)$, e.g., nearly a half of these classifiers classify (x_i, y_i) as a positive example while the remaining classifiers classify it as a negative one, it is considered to be a noise and removed from the training set. In this way, a refined training set D'_{tr} is obtained and the quality of the original training set is accordingly improved. When the original training set contains a relative high proportion of noises, the generated classifiers $\{C_t\}_{t=1}^{T}$ may be not accurate enough, in this case, some misdetected noisy examples (in fact, correctly labeled examples) will be removed, leading to a degraded generalization performance of the classifiers learned on the refined training set.

(2) Another category of approaches tries to improve the robustness of AdaBoost by modifying its loss function or weight updating scheme.

To enhance the noise tolerance of AdaBoost, Domingo et al., [23] proposed a modification of AdaBoost in which the weights of training examples are bounded by their initial values. Servedio [24] designed a boosting algorithm that generates smooth weight distributions on the training set and prevents from assigning too large weight to any single training example. Although the two approaches both try to reduce the harmful effect of noisy examples by modifying the weighting scheme, they do not explicitly define which and how many examples in the original training set should be considered as mislabeled noises. Furthermore, empirical studies have shown that they can not achieve significant performance improvement over AdaBoost in the training scenario with mislabeled noises [23,25].

Gao et al., [25] presented an edited AdaBoost algorithm named EAdaBoost that utilizes the weighted kNN technique. In each iteration of EAdaBoost, a weighted kNN algorithm is adopted to divide training set D_{tr} into two parts, i.e., the noisy examples and the non-noisy examples. When updating the weights of training examples, all the misclassified examples in this iteration are assigned larger weights as in classical AdaBoost, in addition, the weights of all the examples detected as noisy ones are set to zero. In this way, it is expected that the influence of these so-detected noisy examples would be eliminated in the next iteration and thus the quality of the corresponding training set is improved. However, when calculating the weighted training error of the classifier generated in each iteration, EAdaBoost simply counts the weights of the misclassified examples without respectively considering the cases of noisy examples and non-noisy ones.

In the work of Cao et al., [26], they pointed out that AdaBoost assigns larger weights to misclassified training examples but ignores the fact that there may exist some mislabeled noisy examples, in this case, both the misclassified non-noisy examples and the misclassified noisy ones are assigned larger weights. As a result, they will have a greater chance to be selected into the training set of the next base classifier, which can make AdaBoost focus on learning these misclassified noisy examples and lead to overfitting. To solve this problem, Cao et al., [26] proposed a noise-detection based AdaBoost algorithm, i.e., ND_AdaBoost, whose basic idea is as follows. After the first classifier C_1 is trained on the original training set D_{tr}, in each of the following iterations, a noise-detection function is first utilized to divide all the training examples into two subsets: (1) the non-noisy examples and (2) the mislabeled noisy ones. Then, the detected non-noisy examples are processed in the same way as that in the classical AdaBoost, i.e., the misclassified examples are assigned larger weights while the correctly classified ones are assigned lower weights; the detected noisy examples are processed in the opposite way, i.e., the misclassified examples are assigned lower weights while the correctly classified ones are assigned larger weights, which makes the next classifier focus on learning the misclassified non-noisy examples and the correctly classified noisy ones. Doing so they expect that the misclassified non-noisy examples would be correctly classified and the correctly classified noisy examples would be misclassified in the next iteration. Their experimental results demonstrated that ND_AdaBoost is more robust than AdaBoost on training sets with mislabeled noises.

However, ND_AdaBoost can only enhance the robustness of the classical two-class AdaBoost algorithm and is hardly applied to the multi-class classification scenario, which is mainly due to the following reasons. (1) Its two-class loss function is not appropriate to the multi-class problems and its accuracy requirement for the individual base classifiers, i.e., greater than 0.5, is too strong to be met in the multi-class scenario. (2) ND_AdaBoost still has the tendency of overfitting as it increases the weights of correctly classified noisy examples in each iteration, which could make it focus on learning these noisy examples in the subsequent iterations.

3. ND_AdaBoost and SAMME

In this section, to facilitate the illustration of our algorithm, we first introduce the noise-detection based AdaBoost algorithm ND_AdaBoost that has been proposed to improve the robustness of AdaBoost in the two-class classification scenario, then review the multi-class AdaBoost algorithm SAMME that serves as the foundation of our algorithm.

3.1. Noise-detection based AdaBoost algorithm ND_AdaBoost

Now, we use the symbols adopted in this paper to briefly introduce the noise-detection based AdaBoost algorithm ND_AdaBoost [26] for the two-class classification scenario.

In the *iter*th ($1 \le iter \le I$) iteration of ND_AdaBoost, after the base classifier g_t ($1 \le t \le T,\ t \le iter$) is generated, it is used to classify the training set $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. Then, according to the classification results $\{g_t(x_i)\}_{i=1}^{N}$ of g_t, a noise detection function *NDF* is utilized to determine the noise label $\phi_{iter}(x_i)$ of each training example $(x_i,\ y_i)$ $(i = 1, 2, \ldots, N)$, where the details of *NDF* are given in subsection 3.1.1. If example $(x_i,\ y_i)$ is deemed to be a mislabeled noise in this iteration, its noise label is set to -1: $\phi_{iter}(x_i) = -1$; otherwise, its noise label is 1: $\phi_{iter}(x_i) = 1$. After the weighted training error err_t and the weight α_t of classifier g_t are calculated using the corresponding formulae, if the training error err_t satisfies the accuracy requirement (i.e., $err_t < 0.5$), ND_AdaBoost updates the weights of training examples in the following way. For the ith $(i = 1, 2, \ldots, N)$ example $(x_i,\ y_i)$, (1) if it is detected as a non-noisy example, i.e., $\phi_{iter}(x_i) = 1$, it will be processed in the same way as that in classical AdaBoost: its weight is increased (decreased) when it is misclassified (correctly classified); (2) if $(x_i,\ y_i)$ is detected as a noisy example, i.e., $\phi_{iter}(x_i) = -1$, it will be processed in the opposite way: its weight is decreased (increased) when it is misclassified (correctly classified). In this way, it is expected that the next classifier g_{t+1} could focus on learning the misclassified non-noisy examples and the correctly classified noisy examples such that the misclassified non-noisy examples would be correctly classified while the correctly classified noisy examples would be misclassified in the subsequent iteration. The main steps of ND_AdaBoost are presented in Algorithm 1 .

3.1.1. Noise-detection function

As mentioned above, in each iteration of ND_AdaBoost, a noise-detection function *NDF* is utilized to determine the noise label of each training example. In this subsection, we introduce the *NDF* utilized by Cao et al., [26], which has been shown to have good performance.

In the *iter*th ($1 \le iter \le I$) iteration of ND_AdaBoost, after classifier g_t ($1 \le t \le T,\ t \le iter$) is generated, the *NDF* determines the noise label $\phi_{iter}(x_i)$ of example $(x_i,\ y_i)$ $(i = 1, 2, \ldots, N)$ by utilizing its k nearest neighbors $\{(x_{ij}, y_{ij})\}_{j=1}^{k}$ $(ij \in \{1, 2, \ldots, N\}, ij \neq i)$ and the classification error rate of g_t in these neighbors.

Now, we use a simple instance to illustrate the working mechanism of the *NDF*. Suppose we have a training set D_{tr} in which each example is described using two numerical features, and there are three classes 0,1,2 in D_{tr}. The training example $S1 = (0.75, 1.00, 0)$ under consideration is denoted using the black circle in Fig. 1, where '(0)' represents that the class label of $S1$ is 0, while the red squares and blue diamonds denote different training examples. First, *NDF* finds the k (e.g., $k = 5$) nearest neighbors of $S1$ in D_{tr}, and it is easy to see that examples $S2$, $S3$, $S4$, $S5$, $S6$ denoted by the five red squares in Fig. 1 are the five nearest neighbors of $S1$, i.e., $neighbors(S1) = \{S2, S3, S4, S5, S6\}$. Then, *NDF* calculates the classification error rate of classifier g_t in the neighborhood and views it as the probability of $S1$ being a mislabeled noisy example. Suppose the true class labels of $S2$, $S3$, $S4$, $S5$, $S6$ are respectively 0,1,2,0,1, the classification results of g_t on the five examples are 0,2,2,1,1, respectively. In this case, g_t's classification error rate in $neighbors(S1)$ is $\frac{2}{5} = 0.4$, i.e., the probability of $S1$ being a mislabeled noise is 0.4 in this iteration: $\mu_{iter}(S1) = 0.4$. The probabilities of all the other training examples are calculated in the same way. Next, the average value $\overline{\mu_{iter}}$ over the calculated probabilities of all the training

```
Input: Training data set: $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where
       $x_i \in \mathbb{R}^d$ $(d > 1), y_i \in Y = \{-1, 1\}, i = 1, 2, \ldots, N$;  Base
       classifier learning algorithm: Learn;  Ensemble size:
       $T$;  Maximum iteration time: $I$.
Output: Ensemble classifier: $f^{(T')} = \sum_{t=1}^{T'} \alpha_t g_t$, where $T'$ is
       the actual number of generated base classifiers.
 • Initialization
1  Weights of training examples: $w_i^1 = 1/N$, $i = 1, 2, \ldots, N$;
   $t = 0$;  $iter = 0$.
 • Iteration process
2  while $(t < T)$ and $(iter < I)$ do
3      $t = t + 1$; $iter = iter + 1$;
4      Train a base classifier $g_t$ on the training set $D_{tr}$ with
       weight distribution $\{w_i^{iter}\}_{i=1}^N$ :  $g_t = Learn(D_{tr}, \{w_i^{iter}\}_{i=1}^N)$;
5      Apply classifier $g_t$ to classify training set $D_{tr}$;
6      Use the noise-detection function NDF to determine the
       noise label $\phi_{iter}(x_i)$ of training example $(x_i, y_i)$,
       $i = 1, 2, \ldots, N$;
7      Calculate the weighted training error of $g_t$:
       $err_t = \sum_{y_i g_t(x_i)\phi_{iter}(x_i)=-1} w_i^{iter}$;
8      Compute the weight of classifier $g_t : \alpha_t = \frac{1}{2} ln(\frac{1-err_t}{err_t})$;
9      if $err_t \geq 0.5$ or $err_t = 0$ then
10         Generate uniform weight distribution: $w_i^{iter+1} = 1/N$,
           $i = 1, 2, \ldots, N$;
11         Re-train the $t$th classifier in the next iteration:
           $t = t - 1$;
12         break;
13     end
14     Update the weights of training examples:
       $w_i^{iter+1} = w_i^{iter} exp\big(-\alpha_t y_i g_t(x_i)\phi_{iter}(x_i)\big)$,  $i = 1, 2, \ldots, N$;
15     Normalize the weights of training examples:
       $w_i^{iter+1} = \frac{w_i^{iter+1}}{\sum_{j=1}^N w_j^{iter+1}}$,  $i = 1, 2, \ldots, N$;
16 end
```

Algorithm 1: The main steps of the ND_AdaBoost algorithm.

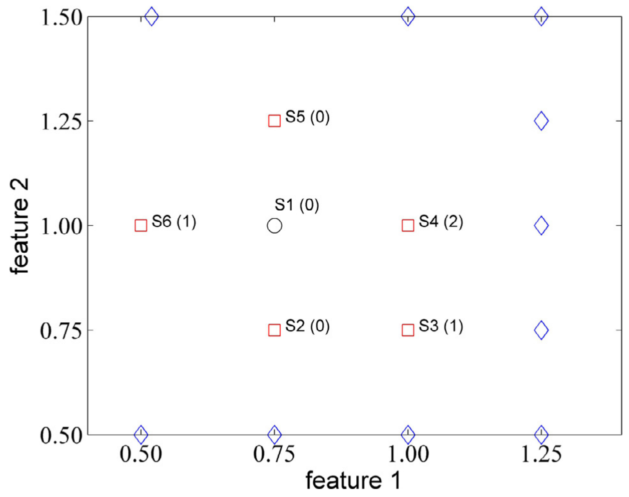


Fig. 1. A simple instance illustrating a training example's neighbors.

examples is obtained: $\overline{\mu_{iter}} = \frac{1}{N}\sum_{i=1}^N \mu_{iter}(x_i, y_i)$. Finally, we determine the noise label of each example. For the ith $(i = 1, 2, \ldots, N)$ example $(x_i,\ y_i)$, if its probability is greater than the average value: $\mu_{iter}(x_i, y_i) > \overline{\mu_{iter}}$, it is considered as a mislabeled noise and its noise label is set to -1: $\phi_{iter}(x_i) = -1$; otherwise, it is considered as a non-noisy example, i.e., $\phi_{iter}(x_i) = 1$. The general procedure of the *NDF* is formally demonstrated in Algorithm 2

```
Input: Training data set: $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$;
       Classification results of classifier $g_t$ on training set $D_{tr}$:
       $\{g_t(x_i)\}_{i=1}^N$;  Size of neighborhood: $k$.
Output: The detected noise labels of all the training
       examples in the $iter$th iteration: $\{\phi_{iter}(x_i)\}_{i=1}^N$.
 • Determination process
1  for each example $(x_i, y_i)$ in $D_{tr}$ do
2      Find its $k$ nearest neighbors in $D_{tr}$: $neighbors(x_i, y_i)$ =
       $\{(x_{ij}, y_{ij})\}_{j=1}^k$, where $ij \in \{1, 2, \ldots, N\}, ij \neq i$, and the
       Euclidean distance is used as the distance metric;
3      Calculate the probability of example $(x_i, y_i)$ being a
       mislabeled noise, i.e., the classification error rate of
       classifier $g_t$ in its $k$ nearest neighbors:
       $\mu_{iter}(x_i, y_i) = \frac{1}{k}\sum_{j=1}^k I(g_t(x_{ij}) \neq y_{ij})$;
4  end
5  Calculate the average value over the obtained probabilities of
   all the training examples: $\overline{\mu_{iter}} = \frac{1}{N}\sum_{i=1}^N \mu_{iter}(x_i, y_i)$;
6  for each example $(x_i, y_i)$ in $D_{tr}$ do
7      if $\mu_{iter}(x_i, y_i) > \overline{\mu_{iter}}$ then
8          Its noise label is set to $-1$: $\phi_{iter}(x_i) = -1$, put it into
           the noisy example set NS: $NS = NS \cup \{(x_i, y_i)\}$;
9      end
10     else if $\mu_{iter}(x_i, y_i) \leq \overline{\mu_{iter}}$ then
11         Its noise label is set to 1: $\phi_{iter}(x_i) = 1$, put it into the
           non-noisy example set NNS: $NNS = NNS \cup \{(x_i, y_i)\}$;
12     end
13 end
```

Algorithm 2: The procedure of the noise-detection function *NDF*.

3.2. Multi-class AdaBoost algorithm SAMME

Although there already exist many multi-class AdaBoost algorithms, such as SAMME[21], AdaBoost.Cost[27], AdaBoost.M1[28], AdaBoost.MO[29], AdaBoost.MH[30], AdaBoost.MR[30], AdaBoost.HM[31], etc., we choose Zhu et al.'s multi-class AdaBoost algorithm SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function)[21] as the foundation of our algorithm. This is due to that SAMME is a very effective and popular multi-class AdaBoost algorithm (has been cited 309 times since 2009) and can be *directly* applied to the multi-class classification case without reducing it to multiple two-class problems. In this section, we first introduce the multi-class exponential loss function of SAMME, then give its main steps.

3.2.1. Multi-class exponential loss function

In the multi-class classification scenario, the class label y_i of example $(x_i,\ y_i)$ in training set $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ $(x_i \in \mathbb{R}^d, y_i \in Y = \{1, 2, \ldots, K\}, i = 1, 2, \ldots, N,\ K \geq 3$ is the number of distinct class labels) is encoded by a K dimensional vector: $Y_i = (Y_{i1}, Y_{i2}, \ldots, Y_{iK})$, where each element in Y_i is defined as follows:

$$Y_{ij} = \begin{cases} 1 & if \quad y_i = j, \\ -\frac{1}{K-1} & if \quad y_i \neq j. \end{cases} \tag{1}$$

It is easy to see that the K elements in Y_i satisfy: $Y_{i1} + Y_{i2} + \cdots + Y_{iK} = 0$.

Definition 1 (The multi-class exponential loss function)**.** Using the above encoding strategy, the multi-class exponential loss function is defined as follows:

$$L(Y,\mathbf{f}) = \sum_{i=1}^{N} \exp\Big(-\frac{1}{K}\big(Y_i^T \mathbf{f}^{(T)}(x_i)\big)\Big) = \sum_{i=1}^{N} \exp\Big(-\frac{1}{K} Y_i^T\big(\mathbf{f}^{(T-1)}(x_i) + \alpha_T \mathbf{g}_T(x_i)\big)\Big), \quad (2)$$

Where $\mathbf{f}^{(T)}(x_i) = (f_1(x_i), \ldots, f_K(x_i))$ is the K dimensional vector corresponding to the class label $f^{(T)}(x_i)$ predicted by the ensemble classifier $f^{(T)}(x)$, which is obtained by the weighted combination of the first T $(T \geq 1)$ base classifiers: $f^{(T)}(x) = \sum_{t=1}^{T} \alpha_t g_t(x) = f^{(T-1)}(x) + \alpha_T g_T(x)$, ($g_T(x)$ is the Tth base classifier and α_T is its weight); $Y_i^T = (Y_{i1}, \ldots, Y_{iK})^T$ is the transpose of the K dimensional vector corresponding to the true class label y_i of example (x_i, y_i).

Now, we illustrate the above symbols through a concrete instance to make them easier to understand. Suppose there are three classes $Y = \{1, 2, 3\}$ (i.e., $K = 3$) in training set $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the ensemble classifier $f^{(T)}(x)$ is composed by five (i.e.,$T = 5$) base classifiers (e.g., five decision trees or five neural networks) $\{g_1, g_2, \ldots, g_5\}$ and their corresponding weights $\{\alpha_1, \alpha_2, \ldots, \alpha_5\}$. For training example ($x_i$, y_i) $(1 \leq i \leq N)$, suppose the ensemble $f^{(5)}(x)$'s classification result is 3, i.e., $f^{(5)}(x_i) = 3$, and the example's true class label is 1: $y_i = 1$. In this case, according to Eq. (1), the K dimensional vectors corresponding to $f^{(5)}(x_i)$ and y_i are $\mathbf{f}^{(5)}(x_i) = (-\frac{1}{2}, -\frac{1}{2}, 1)$ and $Y_i = (1, -\frac{1}{2}, -\frac{1}{2})$, respectively. We have $Y_i^T \mathbf{f}^{(5)}(x_i) = (1, -\frac{1}{2}, -\frac{1}{2})^T \cdot (-\frac{1}{2}, -\frac{1}{2}, 1) = -\frac{1}{2} + \frac{1}{4} - \frac{1}{2} = -\frac{3}{4} = -\frac{K}{(K-1)^2} < 0$. Thus, the classification loss on this training example is $L(Y_i, \mathbf{f}_i^{(5)}) = -\frac{1}{K} Y_i^T \mathbf{f}^{(5)}(x_i) = -\frac{1}{3} \times (-\frac{3}{4}) = \frac{1}{4} = \frac{1}{(K-1)^2} > 0$. Similarly, for training example (x_i, y_i) $(1 \leq i \leq N)$, if the predicted label $f^{(5)}(x_i)$ equals to its true label y_i, e.g., $f^{(5)}(x_i) = y_i = 1$, we have $Y_i^T \mathbf{f}^{(5)}(x_i) = (1, -\frac{1}{2}, -\frac{1}{2})^T \cdot (1, -\frac{1}{2}, -\frac{1}{2}) = \frac{3}{2} = \frac{K}{K-1} > 0$, and thus the corresponding classification loss is: $L(Y_i, \mathbf{f}_i^{(5)}) = -\frac{1}{K} Y_i^T \cdot \mathbf{f}^{(5)}(x_i) = -\frac{1}{3} \times \frac{3}{2} = -\frac{1}{2} = -\frac{1}{K-1} < 0$.

From the above analysis, it can be easily inferred that a correct classification decreases the loss function in Eq. (2) while an incorrect classification increases the loss function. Naturally, it is expected to find the weight α_T $(\alpha_T \in \mathbb{R}, \alpha_T > 0)$ of base classifier $g_T(x)$ $(T \geq 1)$ such that the multi-class exponential loss function $L(Y, \mathbf{f})$ on training set D_{tr} is minimized:

$$(\alpha_T, g_T) = \mathrm{argmin}_{\alpha,g} \sum_{i=1}^{N} \exp\Big(-\frac{1}{K} Y_i^T\big(\mathbf{f}^{(T-1)}(x_i) + \alpha_T \mathbf{g}_T(x_i)\big)\Big) = \mathrm{argmin}_{\alpha,g} \sum_{i=1}^{N} w_i^{iter} \exp\Big(-\frac{1}{K}\alpha_T Y_i^T \mathbf{g}_T(x_i)\Big), \quad (3)$$

Where $w_i^{iter} = exp\big(-\frac{1}{K} Y_i^T \mathbf{f}^{(T-1)}(x_i)\big)$ is the weight of example (x_i, y_i) $(1 \leq i \leq N)$ in the *iter*th iteration in which the Tth classifier g_T is generated ($T \leq iter$, this is because some classifiers generated in the previous iterations may not satisfy the accuracy requirement and thus are regenerated in the subsequent iterations). It is easy to see that w_i^{iter} is a constant for the Tth base classifier $g_T(x)$ and its weight α_T. Zhu et al., [21] solved this minimization problem by proving a lemma and got the optimal weight α_T of the Tth base classifier $g_T(x)$, leading to the multi-class AdaBoost algorithm SAMME.

3.2.2. The main steps of SAMME

The main steps of multi-class AdaBoost algorithm SAMME are formally depicted in Algorithm 3 . It can be seen from Algorithm 3 that the differences between this multi-class AdaBoost algorithm and the classical two-class AdaBoost algorithm mainly lie in two aspects. (1) The formulae for calculating the weights of base classifiers are different. When calculating the weight of classifier g_t $(1 \leq t \leq T)$, there is an extra term $ln(K-1)$ in SAMME, and it has been indicated by Zhu et al., [21] that this term is not artificial but follows naturally from the multi-class generalization of the exponential loss function in the two-class case. Obviously, when $K = 2, ln(K-1) = ln1 = 0$, it reduces to the case of two-class AdaBoost; (2) The accuracy requirements for base classifiers are different. In SAMME, the accuracy of each trained classifier only needs to be better than random guessing (i.e., $1/K$) instead of 0.5 that is too hard to be met in the multi-class scenario. That is, only when the weighted training error of classifier g_t is greater than or equal to $(K-1)/K$ can it be rejected and re-trained in the next iteration.

```
Input: Training set: $D_{tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where
       $x_i \in \mathbb{R}^d$ $(d > 1), y_i \in Y = \{1, 2, \ldots, K\}, i = 1, 2, \ldots, N$,
       and $K$ $(K \geq 3)$ is the number of distinct class labels;
       Base classifier learning algorithm: Learn; Ensemble
       size: $T$; Maximum iteration time: $I$.
Output: Ensemble classifier: $f^{(T')} = \sum_{t=1}^{T'} \alpha_t g_t$, where $T'$ is
        the actual number of generated base classifiers.
 • Initialization
1 Weights of training examples: $w_i^1 = 1/N,\ i = 1, 2, \ldots, N;\ t = 0$;
  $iter = 0$.
 • Iteration process
2 while $(t < T)$ and $(iter < I)$ do
3     $t = t + 1;\ iter = iter + 1$;
4     Train a base classifier $g_t$ on the training set $D_{tr}$ with
      weight distribution $\{w_i^{iter}\}_{i=1}^N : g_t = Learn(D_{tr}, \{w_i^{iter}\}_{i=1}^N)$;
5     Apply classifier $g_t$ to classify training set $D_{tr}$;
6     Calculate the weighted training error of
      $g_t : err_t = \sum_{g_t(x_i) \neq y_i} w_i^{iter}$;
7     Compute the weight of classifier
      $g_t : \alpha_t = ln(\frac{1-err_t}{err_t}) + ln(K-1)$;
8     if $err_t \geq (K-1)/K$ then
9         Generate uniform weight distribution: $w_i^{iter+1} = 1/N$,
          $i = 1, 2, \ldots, N$;
10        Re-train the $t$th classifier in the next iteration:
          $t = t - 1$;
11        break;
12    end
13    Update the weights of training examples:
      $w_i^{iter+1} = w_i^{iter} exp\big(\alpha_t I(y_i \neq g_t(x_i))\big), \quad i = 1, 2, \ldots, N$;
14    Normalize the weights of training examples:
      $w_i^{iter+1} = \frac{w_i^{iter+1}}{\sum_{j=1}^N w_j^{iter+1}}, \quad i = 1, 2, \ldots, N$;
15 end
```

Algorithm 3: The main steps of SAMME.

It should be noted that the maximum iteration time I is used to guarantee that the algorithm can terminate within finite execution time when the extreme case occurs, i.e., most of the generated classifiers do not satisfy the accuracy requirement ($> 1/K$). Therefore, when T classifiers are generated or the maximum iteration time I is exceeded, SAMME terminates and the final ensemble is the weighted combination of all the generated classifiers, i.e., $f(x) = \mathrm{argmax}_{y \in Y} \sum_{t=1,2,\ldots,T'} \alpha_t I(g_t(x) = y)$, where T' is the actual number of all the generated classifiers.

4. A robust multi-class AdaBoost algorithm

In this section, we first illustrate the basic idea of our robust multi-class AdaBoost algorithm Rob_MulAda, then introduce the designed noise-detection based multi-class loss function, and finally give the main steps of our algorithm.

4.1. The basic idea of our algorithm

Inspired by Cao et al.'s work for the classical two-class AdaBoost, i.e., ND_AdaBoost [26] introduced in Section 3.1, we propose a robust multi-class AdaBoost algorithm named Rob_MulAda, which follows the basic architecture of the multi-class AdaBoost algorithm SAMME [21]. In our algorithm, a noise-detection based multi-class loss function is formally designed and a new weight updating scheme is presented.

The basic idea of Rob_MulAda is as follows. In the *iter*th ($1 \le iter \le I$) iteration, after the generated classifier g_t ($t \le iter$, this is because some classifiers generated in previous iterations may not satisfy the accuracy requirement and thus are regenerated in subsequent iterations) makes predictions for training set D_{tr}, we utilize the noise-detection function *NDF* introduced in subsection 3.1.1 to determine the noise label $\phi_{iter}(x_i)$ of each example $(x_i,\ y_i)$, $i = 1, 2, \ldots, N$. If $\phi_{iter}(x_i) = -1(1)$, then $(x_i,\ y_i)$ is considered as a mislabeled noisy (non-noisy) example in this iteration, and thus we can divide all the training examples into two subsets according to their noise labels: (1). the non-noisy examples $NNS = \{(x_i, y_i)|\phi_{iter}(x_i) = 1, i = 1, 2, \ldots, N\}$ and (2). the noisy examples $NS = \{(x_i, y_i)|\phi_{iter}(x_i) = -1, i = 1, 2, \ldots, N\}$. Then, the weighted training error err_t of classifier g_t is determined in this way: we not only count the weights of misclassified non-noisy examples but also the weights of correctly classified noisy examples, which follows naturally from the intuition that the non-noisy examples should be correctly classified while the mislabeled noisy examples be misclassified. Therefore, classifier g_t is penalized if it misclassifies non-noisy examples or correctly classifies noisy examples. We believe that this weighted training error can more objectively reflect the classifier's generalization performance. Next, the weight α_t of classifier g_t is calculated. After that, if the training error err_t of classifier g_t satisfies the requirement, i.e., $err_t < (K-1)/K$, then the weights of all the training examples are updated *according to our weight updating scheme*: for the misclassified non-noisy examples, their weights are increased as done in the classical two-class AdaBoost algorithm and the multi-class AdaBoost algorithm SAMME introduced in Section 3.2; for these correctly classified noisy examples, their weights are set to zero rather than being increased as in the work of Cao et al., [26].

Now we adopt a simple instance to illustrate the potential advantage of our weight updating scheme, which is implemented by analyzing its impact on true noisy examples and examples erroneously detected as noises. For each training example $(x_i,\ y_i)$ $(i = 1, 2, \ldots, N)$ that is detected as mislabeled noise in the *iter*th iteration, i.e., $\phi_{iter}(x_i) = -1$, we consider the following two cases. (1). Example $(x_i,\ y_i)$ is *correctly* detected as a noise by the *NDF* and is indeed a mislabeled noise. In this case, if $(x_i,\ y_i)$ is correctly classified by the current classifier g_t (i.e., $g_t(x_i) = y_i$), our weight updating scheme assigns weight zero to $(x_i,\ y_i)$ such that all the noisy examples like this one would not be emphasized for learning by the next classifier and thus avoids overfitting them. However, the weight updating scheme utilized in ND_AdaBoost assigns larger weight to $(x_i,\ y_i)$, in this way, all the noisy examples like this one would have a larger possibility to be selected into the training set of the next classifier, naturally overfitting is very likely to happen; (2). Example $(x_i,\ y_i)$ is *erroneously* detected as a noise by the *NDF* and is actually a non-noisy example. In this case, if $(x_i,\ y_i)$ is correctly classified by g_t, our weight updating scheme assigns zero to its weight while the one utilized in ND_AdaBoost assigns a larger weight. As now $(x_i,\ y_i)$ is actually a normal (non-noisy) example, our weight updating scheme also complies with the principle of the classical AdaBoost algorithm: the correctly classified normal examples will not be emphasized in the subsequent iteration, but the scheme used by ND_AdaBoost still emphasizes these easy-to-learn training examples. Generally, case (1) is more likely to happen than case (2). To sum up, in the two cases, i.e., for both *correctly detected* and *erroneously detected noisy examples*, our weight updating scheme could be more effective than the one used in ND_AdaBoost. Therefore, we infer that our weight updating scheme should have stronger robustness against mislabeled noises.

4.2. Noise-detection based multi-class loss function

Based on the multi-class loss function introduced in Section 3.2 and the detected noise label $\phi_{iter}(x_i)$ of each training example $(x_i,\ y_i)$ $(i = 1, 2, \ldots, N)$, we formally design a noise-detection based multi-class loss function which is a key component of our algorithm.

Definition 2 (The designed multi-class loss function)**.** The noise-detection based multi-class loss function used in our algorithm is formally represented as:

$$\bar{L}(Y, \mathbf{f}) = \sum_{i=1}^{N} \overline{w_i}^{iter} \exp\left(-\frac{1}{K}\alpha_T Y_i^T \mathbf{g}_T(x_i)\phi_{iter}(x_i)\right), \tag{4}$$

Where $\overline{w_i}^{iter} = \exp\left(-\frac{1}{K} Y_i^T \mathbf{f}^{(T-1)}(x_i)\right)$ is the weight of example (x_i, y_i) $(i = 1, 2, \ldots, N)$ and a constant for the Tth ($T \ge 1$) base classifier $g_T(x)$ and its weight α_T, *iter* in $\overline{w_i}^{iter}$ denotes the *iter*th ($1 \le iter \le I$, $iter \ge T$) iteration in which base classifier g_T is generated; similarly, $\mathbf{f}^{(T-1)}(x_i)$ is the K dimensional vector corresponding to the class label of example $(x_i,\ y_i)$ predicted by the ensemble classifier $f^{(T-1)}(x)$, which is obtained by the weighted combination of the first $T-1$ classifiers, i.e., $f^{(T-1)} = \sum_{t=1}^{T-1} \alpha_t g_t(x)$; the noise label $\phi_{iter}(x_i)$ is used to indicate whether example $(x_i,\ y_i)$ is a mislabeled noise or not in the *iter*th iteration that generates classifier g_T. As already introduced in subsection 3.1.1, $\phi_{iter}(x_i) \in \{-1, 1\}$, $\phi_{iter}(x_i) = -1$ represents that example $(x_i,\ y_i)$ is considered as a noise; otherwise, it is considered as a non-noisy example.

From the designed loss function in Eq. (4), it can be seen that classifier g_T is penalized if it misclassifies a non-noisy example (in this case, $Y_i^T \mathbf{g}_T(x_i) < 0, \phi_{iter}(x_i) = 1, -\frac{1}{K}\alpha_T Y_i^T \mathbf{g}_T(x_i)\phi_{iter}(x_i) > 0$) or correctly classifies a noisy example (in this case, $Y_i^T \mathbf{g}_T(x_i) > 0, \phi_{iter}(x_i) = -1, -\frac{1}{K}\alpha_T Y_i^T \mathbf{g}_T(x_i)\phi_{iter}(x_i) > 0$). Therefore, our aim is to minimize this designed loss function, which requires to find the optimal weight α_T of the Tth ($T \ge 1$) base classifier g_T. We solve this optimization problem by proving the following proposition.

Proposition 1. To minimize the noise-detection based multi-class loss function in Eq. (4), the weight α_T of the Tth ($T \ge 1$) base classifier g_T should be: $\alpha_T = \frac{(K-1)^2}{K}\left(ln\left(\frac{1-err_T}{err_T}\right) + ln(K-1)\right)$, where $err_T = \sum_{Y_i^T \mathbf{g}_T(x_i)\phi_{iter}(x_i) < 0} \overline{w_i}^{iter}$ is the weighted training error of the Tth classifier g_T.

Proof. The loss function in Eq. (4) can be divided into parts (A) and (B) depending on the classification results of base classifier g_T:

$$\bar{L}(Y, \mathbf{f}) = \underbrace{\sum_{Y_i^T \mathbf{g}_T > 0} \overline{w_i}^{iter} \exp\left(-\frac{\alpha_T}{K-1}\phi_{iter}(x_i)\right)}_{(A)} + \underbrace{\sum_{Y_i^T \mathbf{g}_T < 0} \overline{w_i}^{iter} \exp\left(\frac{\alpha_T}{(K-1)^2}\phi_{iter}(x_i)\right)}_{(B)}, \tag{5}$$

Where $\mathbf{g}_T$ is the abbreviation of $\mathbf{g}_T(x_i)$, Eq. (5) is obtained due to that when $Y_i = \mathbf{g}_T$, we have $Y_i^T\mathbf{g}_T = \frac{K}{K-1} > 0$, and when $Y_i \neq \mathbf{g}_T$, we have $Y_i^T\mathbf{g}_T = -\frac{K}{(K-1)^2} < 0$. Considering the concrete noise label of each example, (A) and (B) can be further divided into two parts respectively as shown in Eq. (6),

$$\begin{aligned}\bar{L}(Y,\mathbf{f}) = &\underbrace{\sum_{Y_i^T\mathbf{g}_T>0,\phi_{iter}=+1} \overline{w_i}^{iter}\exp\left(-\frac{\alpha_T}{K-1}\right)}_{(A1)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T>0,\phi_{iter}=-1} \overline{w_i}^{iter}\exp\left(\frac{\alpha_T}{K-1}\right)}_{(A2)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T<0,\phi_{iter}=-1} \overline{w_i}^{iter}\exp\left(-\frac{\alpha_T}{(K-1)^2}\right)}_{(B1)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T<0,\phi_{iter}=+1} \overline{w_i}^{iter}\exp\left(\frac{\alpha_T}{(K-1)^2}\right)}_{(B2)},\end{aligned} \tag{6}$$

Where ϕ_{iter} is the abbreviation of $\phi_{iter}(x_i)$, by adapting the denominators in (A1) and (B2), we get an upper bound $\widetilde{L}(Y,\mathbf{f})$ (i.e., $\bar{L}(Y,\mathbf{f}) \leq \widetilde{L}(Y,\mathbf{f})$) of Eq. (6) as shown in Eq. (7):

$$\begin{aligned}\widetilde{L}(Y,\mathbf{f}) = &\underbrace{\sum_{Y_i^T\mathbf{g}_T>0,\phi_{iter}=+1} \overline{w_i}^{iter}\exp\left(-\frac{\alpha_T}{(K-1)^2}\right)}_{(A1)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T>0,\phi_{iter}=-1} \overline{w_i}^{iter}\exp\left(\frac{\alpha_T}{K-1}\right)}_{(A2)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T<0,\phi_{iter}=-1} \overline{w_i}^{iter}\exp\left(-\frac{\alpha_T}{(K-1)^2}\right)}_{(B1)}\\ &+\underbrace{\sum_{Y_i^T\mathbf{g}_T<0,\phi_{iter}=+1} \overline{w_i}^{iter}\exp\left(\frac{\alpha_T}{K-1}\right)}_{(B2)},\end{aligned} \tag{7}$$

By merging (A1) with (B1), (A2) with (B2), respectively, Eq. (7) is transformed to Eq. (8):

$$\begin{aligned}\widetilde{L}(Y,\mathbf{f}) = &\sum_{Y_i^T\mathbf{g}_T\phi_{iter}>0} \overline{w_i}^{iter}\exp\left(-\frac{\alpha_T}{(K-1)^2}\right)\\ &+\sum_{Y_i^T\mathbf{g}_T\phi_{iter}<0} \overline{w_i}^{iter}\exp\left(\frac{\alpha_T}{K-1}\right),\end{aligned} \tag{8}$$

Let $err_T = \sum_{Y_i^T\mathbf{g}_T\phi_{iter}<0} \overline{w_i}^{iter}$ be the weighted training error of the Tth classifer g_T, then Eq. (8) is transformed to Eq. (9):

$$\widetilde{L}(Y,\mathbf{f}) = (1-err_T)\ \exp\left(-\frac{\alpha_T}{(K-1)^2}\right) + err_T\exp\left(\frac{\alpha_T}{K-1}\right), \tag{9}$$

For Eq. (9), take the derivative with respect to α_T and equal to zero, we have:

$$\frac{\partial\widetilde{L}(Y,\mathbf{f})}{\partial\alpha_T} = -\frac{1-err_T}{(K-1)^2}\exp\left(-\frac{\alpha_T}{(K-1)^2}\right) + \frac{err_T}{K-1}\exp\left(\frac{\alpha_T}{K-1}\right) = 0, \tag{10}$$

The answer to Eq. (10) is:

$$\alpha_T = \frac{(K-1)^2}{K}\left(\ln\left(\frac{1-err_T}{err_T}\right) + \ln(K-1)\right). \tag{11}$$

Thus, the optimal weight of the Tth base classifier g_T is obtained. $\square$

4.3. Main steps of the proposed algorithm

According to the above description, the main steps of the proposed algorithm Rob_MulAda are formally depicted in Algorithm 4 .

It can be seen from Algorithm 4 that, compared with Zhu et al.'s multi-class AdaBoost algorithm SAMME[21], our robust multi-class AdaBoost algorithm Rob_MulAda has the following *unique* characteristics. In the *iter*th ($1 \leq iter \leq I$) iteration: (1) It utilizes the noise detection function *NDF* introduced in subsection 3.1.1 to determine the noise label $\phi_{iter}(x_i)$ of each training example $(x_i,\ y_i)$ $(i = 1, 2, \ldots, N)$ such that it can process the detected noisy and non-noisy examples separately; (2) When calculating the weighted training error err_t of classifier g_t $(1 \leq t \leq T,\ t \leq iter)$ generated in the current iteration, it not only counts the weights of misclassified normal (non-noisy) examples but also the weights of

```
Input: Training set: $D_{tr} = \{(x_1,y_1),\ldots,(x_N,y_N)\}$, where
       $x_i \in \mathbb{R}^d$ $(d > 1)$, $y_i \in Y = \{1,2,\ldots,K\}$, $K$ $(K \geq 3)$ is the
       number of distinct class labels, $i = 1,2,\ldots,N$;  Base
       classifier learning algorithm: $Learn$;  Ensemble size:
       $T$;  Maximum iteration time: $I$.
Output: Ensemble classifier: $f^{(T')} = \sum_{t=1}^{T'} \alpha_t g_t$, where $T'$ is
        the actual number of all the generated classifiers.
• Initialization
1  Weights of training examples: $w_i^1 = 1/N, i = 1,2,\ldots,N$; $t = 0$;
   $iter = 0$.
• Iteration process
2  while $(t < T)$ and $(iter < I)$ do
3      $t = t+1$;  $iter = iter+1$;
4      Train a base classifier $g_t$ on the training set $D_{tr}$ with
       weight distribution $\{w_i^{iter}\}_{i=1}^N$: $g_t = Learn(D_{tr}, \{w_i^{iter}\}_{i=1}^N)$;
5      Apply classifier $g_t$ to classify training set $D_{tr}$;
6      Use the noise-detection function NDF to determine the
       noise label $\phi_{iter}(x_i)$ of each training example
       $(x_i,y_i), i = 1,2,\ldots,N$;
7      Calculate the weighted training error of classifier $g_t$:
       $err_t = \sum_{Y_i^T\mathbf{g}_t\phi_{iter}<0} w_i^{iter}$, where $\mathbf{g}_t$ denotes $\mathbf{g}_t(x_i)$, which is
       the $K$ dimensional vector corresponding to the
       classification result of classifier $g_t$ on $(x_i,y_i)$, and $\phi_{iter}$
       denotes $\phi_{iter}(x_i)$;
8      Compute the weight of classifier $g_t$:
       $\alpha_t = \frac{(K-1)^2}{K}\left(ln(\frac{1-err_t}{err_t}) + ln(K-1)\right)$;
9      if $err_t \geq (K-1)/K$ or $err_t = 0$ then
10         Generate uniform weight distribution on training set
           $D_{tr}$: $w_i^{iter+1} = 1/N$,  $i = 1,2,\ldots,N$;
11         Re-train the $t$th classifier in the next iteration:
           $t = t-1$;
12         break;
13     end
14     Update the weights of training examples:
       $w_i^{iter+1} = w_i^{iter}exp(\alpha_t)$,  $i \in \{j|\phi_{iter}(x_j) = 1, Y_j^T\mathbf{g}_t(x_j) < 0\}$;
       $w_i^{iter+1} = 0$,  $i \in \{j|\phi_{iter}(x_j) = -1, Y_j^T\mathbf{g}_t(x_j) >$
       $0\}$;
15     Normalize the weights of training examples:
       $w_i^{iter+1} = \frac{w_i^{iter+1}}{\sum_{j=1}^N w_j^{iter+1}}$,  $i = 1,2,\ldots,N$;
16 end
```

Algorithm 4: The main steps of our algorithm.

correctly classified noisy examples. However, SAMME only simply counts the weights of misclassified examples without considering whether these examples are noises or not. Therefore, our designed loss function can more objectively reflect classifier g_t's generalization performance; (3) When updating the weights of training examples, Rob_MulAda does not simply increase the weights of misclassified examples as SAMME does, instead it only increases the weights of the misclassified non-noisy examples, which can avoid focusing on the misclassified noisy examples when training classifier g_{t+1} in the subsequent iteration. In addition, it assigns zero to the weights of the correctly classified noisy examples instead of increasing their weights as ND_AdaBoost does, which can avoid focusing on these noisy examples in the subsequent iteration. The presented new weight updating scheme is expected to be helpful in alleviating the harmful influence of mislabeled noises.

4.4. Time complexity analysis

For our Rob_MulAda, it should be pointed out that the *NDF* only needs to find the k nearest neighbors of each training example in the first iteration ($iter = 1$), and in each of the subsequent iterations, it just simply calculates the classification error rate $\mu_{iter}(x_i, y_i)$ of the generated classifier in the ith ($i = 1, 2, \ldots, N$) training example's k nearest neighbors according to the classifier's classification results on D_{tr} obtained in step 5 of Algorithm 4. Therefore, compared with the multi-class AdaBoost algorithm SAMME, the more time consumed by our Rob_MulAda is *mainly* caused by its additional operations conducted for noise detection: (a) the determination of each training example's k nearest neighbors in the first iteration and (b) the calculation of the generated classifier's classification error rate in the ith ($i = 1, 2, \ldots, N$) training example's neighborhood in each iteration. That is, like many other noise-alleviating approaches, our Rob_MulAda also sacrifices some efficiency for the improvement of the robustness.

It can be seen from Algorithm 4 that, in each iteration of Rob_MulAda, the most time is consumed by steps 4 and 6. Although different classifier learning algorithms may have some differences in the time consumed for generating a base classifier, it is always true that the larger the size of training set D_{tr}, the more time Rob_MulAda consumes to complete the above two main steps. Therefore, in one iteration the time complexity is roughly $O(N)$. As the steps executed in each iteration are the same, naturally the time complexity of Rob_MulAda is $O(I'N)$, where I' ($T \le I' \le I$) is the number of iterations actually executed for generating the final ensemble.

5. Experimental study

In this section, we first verify the robustness of our newly-proposed weight updating scheme by directly comparing ND_AdaBoost with its modified version ND_AdaBoost* on 15 two-class classification problems. Then, we compare our robust multi-class AdaBoost algorithm Rob_MulAda with other four closely-related algorithms by conducting experiments on 18 multi-class classification problems such that its effectiveness can be verified. All the adopted two-class and multi-class benchmark data sets are from UCI [32] and Statlog [33] machine learning repositories.

5.1. Verifying the robustness of our weight updating scheme

To verify the robustness of our weight updating scheme, we conduct experiments on 15 two-class benchmark data sets to directly compare our weight updating scheme with the one proposed in ND_AdaBoost [26]. The compared algorithms are ND_AdaBoost introduced in Section 3.1 and its modified version named ND_AdaBoost*, which is purposely designed by just substituting the weight updating scheme in ND_AdaBoost with ours presented in Section 4.1. That is, the *only difference* between ND_AdaBoost and ND_AdaBoost* is that they use different weight updating schemes. For the completeness of comparison, we also include the classical two-class AdaBoost algorithm[15,16] for comparison.

Table 1
Characteristics of the used two-class data sets.

No.	Data set	Examples	Attributes	Classes
1	Australian	690	14	2
2	Breastcancer	683	9	2
3	Chess	3196	36	2
4	Diabetes	768	8	2
5	German	1000	24	2
6	Haberman	306	3	2
7	Heart	270	13	2
8	Indianliver	583	10	2
9	Ionosphere	351	33	2
10	Liverdisorder	345	6	2
11	Monks	556	6	2
12	Sonar	208	60	2
13	Splice	1000	60	2
14	Tictactoe	958	9	2
15	Vertebral	310	6	2

5.1.1. The used two-class data sets

The characteristics of the adopted 15 two-class benchmark data sets are listed in Table 1.

5.1.2. Experimental setting

As decision tree is an effective and unstable classifier learning algorithm, i.e., small changes in the training set can produce large differences in the generated classifiers, thus as its realization, the decision tree CART [34–36] is widely adopted as the base classifier in many ensemble learning algorithms. Here for the three compared boosting algorithms, i.e., AdaBoost, ND_AdaBoost and ND_AdaBoost*, we also adopt CART as the base classifier. All the experiments are carried out in the MatLab software with version 7.11, and the decision tree CART is implemented using the 'classregtree' function contained in the MatLab tool box. For each considered data set D in Table 1, we run five times four-fold cross validation to compare the three algorithms. The size of ensembles generated by the compared algorithms is set to 25, and the main steps of one execution of four-fold cross validation are described in Algorithm 5 .

For step 5 in Algorithm 5, we add mislabeled noisy examples into training set D_{tmp_tr} in this way: randomly select $\lfloor |D_{tmp\_tr}| \cdot noise\_rate \rfloor$ examples from D_{tmp_tr} and then change their class labels to other ones in the label set Y. That is, for each selected example (x_i, y_i), its class label y_i is changed to an arbitrary one $y_i^{'}$ in the remaining class labels: $y_i^{'} \in Y/y_i$.

As the size of the neighborhood is an important parameter in the noise detection function of the two algorithms: ND_AdaBoost, ND_AdaBoost*, to fully utilize the information provided by the neighbors, in our experiments the neighborhood size is set to seven for the two algorithms for fair comparison.

5.1.3. Experimental results and analysis

For each considered data set D and a given mislabeled noise rate *noise_rate*, after five times four-fold cross validation is executed, we obtain the average test accuracies of the three algorithms: $\overline{accu}_{Ada}$, $\overline{accu}_{ND\_Ada}$, $\overline{accu}_{ND\_Ada^*}$. To comprehensively compare the performances of the three algorithms, we consider six different mislabeled noise rates: 5%, 10%, 15%, 20%, 25% and 30%. The concrete experimental results on the 15 data sets are provided in Tables 2, 3, and 4, where 'Ada', 'ND_Ada' and 'ND_Ada*' are the

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$;
       mislabeled noise rate: $noise\_rate$;   the three
       compared algorithms: $AdaBoost$,
       $ND\_AdaBoost$, $ND\_AdaBoost*$.
Output: Average test accuracy of each algorithm over the 4
        folds: $\widetilde{accu}_{Ada}, \widetilde{accu}_{ND\_Ada}, \widetilde{accu}_{ND\_Ada*}$.
1 All the examples in data set $D$ are randomly reordered:
  $D' = \{(x_{jm}, y_{jm})\}_{m=1}^{N}$, where $jm \in \{1, 2, \ldots, N\}, m = 1, 2, \ldots, N$;
2 Divide all the examples in $D'$ into 4 disjoint subsets with
  approximately equal size: $D'_1, D'_2, D'_3, D'_4$;
3 for $i = 1$ to 4 do
4     Use $D'_i$ as the test set $D_{ts}$, and the remaining 3 subsets are
      combined as the training set $D_{tr}$;
5     $D_{tmp\_tr} = D_{tr}$, add a proportion of $noise\_rate$ mislabeled
      noisy examples into training set $D_{tmp\_tr}$ and get a
      contaminated training set $D'_{tmp\_tr}$;
6     Apply the three compared algorithms on $D'_{tmp\_tr}$
      respectively, and then their generated ensembles are
      obtained: $ES_{Ada}, ES_{ND\_Ada}, ES_{ND\_Ada*}$;
7     Use the three ensembles to classify test set $D_{ts}$
      respectively, and their test accuracies in this fold are
      obtained: $accu_{Ada}, accu_{ND\_Ada}, accu_{ND\_Ada*}$.
8 end
9 Compute the average test accuracy of each compared
  algorithm over the 4 folds: $\widetilde{accu}_{Ada}, \widetilde{accu}_{ND\_Ada}, \widetilde{accu}_{ND\_Ada*}$.
```

Algorithm 5: Steps of 4-fold cross validation.

abbreviations of 'AdaBoost', 'ND_AdaBoost' and 'ND_AdaBoost*', respectively.

Table 2 presents the average test accuracies of the three algorithms under noise rates 5% and 10%. Under a specific noise rate, the largest accuracy value in each row is marked in bold face. For instance, under noise rate 5% in Table 2, the largest accuracy value in the first row (i.e., the row corresponding to the obtained results on the Australian data set) is 0.86172 which is obtained by the ND_AdaBoost* algorithm. The last two rows summarize the comparison results of the three algorithms on the total 15 two-class data sets. In the 'Overall result' row, the three numbers in each column give the times that the corresponding algorithm gets the highest average test accuracy, the second highest accuracy and the lowest accuracy, respectively. For instance, the numbers '11-4-0' in the third column denote that, under noise rate 5%, algorithm ND_AdaBoost* achieves the highest average test accuracy on 11 out of the 15 data sets, the second highest accuracy on 4 data sets, and the lowest accuracy on zero data set; The 'Average rank' row presents the average ranks of the three algorithms that are obtained by averaging the ranks of each algorithm over the 15 data sets. The rest of tables in this section can be understood in the same way.

From Tables 2–4, we have the following observations. (1) When the noise rate is low (i.e., $noise\_rate \in (0, 10\%]$), algorithm ND_AdaBoost* significantly improves the generalization performance of AdaBoost, it achieves the largest average test accuracy on about 12 data sets and has the lowest average rank 1.27; ND_AdaBoost can not improve the generalization performance of AdaBoost in some cases, it achieves the largest test accuracy on only 3 data sets and its average rank is much larger than that of ND_AdaBoost*. Therefore, ND_AdaBoost is inferior to ND_AdaBoost*. (2) When the noise rate is moderate (i.e., $noise\_rate \in (10\%, 20\%]$), both ND_AdaBoost* and ND_AdaBoost can well improve the generalization performance of AdaBoost in most cases. However, ND_AdaBoost* is more effective and achieves much higher average test accuracy in most cases, and besides, it has much lower average rank than ND_AdaBoost. (3) When the noise rate is high (i.e., $noise\_rate \in (20\%, 30\%]$), ND_AdaBoost* and ND_AdaBoost still can improve the generalization performance of AdaBoost to varying extents in most cases, and ND_AdaBoost* remains has better overall generalization performance than ND_AdaBoost.

The above observations indicate that, in general, ND_AdaBoost* achieves better generalization performance than ND_AdaBoost under all the considered noise rates. In particular, when the noise rate is low and moderate ($noise\_rate \in (0, 20\%]$), ND_AdaBoost* is obviously much better than ND_AdaBoost. As mentioned above, the *only* difference between ND_AdaBoost and ND_AdaBoost* is that they use different weight updating schemes. Therefore, we can confidently get the conclusion that the superiority of ND_AdaBoost* to ND_AdaBoost is due to that *our weight updating scheme used in ND_AdaBoost* is indeed more robust against mislabeled noises than the one used in ND_AdaBoost.*

5.2. Verifying the effectiveness of Rob_MulAda

In this section, we conduct comparison experiments on 18 multi-class benchmark data sets to verify the effectiveness of our robust multi-class AdaBoost algorithm Rob_MulAda.

5.2.1. The used multi-class data sets

The characteristics of the 18 multi-class benchmark data sets used in our experiments are listed in Table 5.

5.2.2. The compared algorithms

In the experiments, we compare our algorithm with the following four algorithms.

(1) SIN: a single classifier trained on the original training set D_{tr}.
(2) MulAda: the multi-class AdaBoost algorithm SAMME [21] introduced in Section 3.2, which does not take any measure to alleviate the harmful influence of mislabeled noises. To make the name of this algorithm more clear with respect to the names of other compared algorithms, we use 'MulAda' to denote SAMME in the following.
(3) NR_MulAda: this algorithm called Noise-removal based multi-class AdaBoost, which belongs to the first category of approaches introduced in Section 2. NR_MulAda is straightforward, it first detects and removes the mislabeled noisy examples in the original training set D_{tr}, and then applies the multi-class AdaBoost algorithm MulAda on the obtained refined training set D'_{tr}. Concretely, NR_MulAda detects mislabeled noisy examples using the strategy of *kNN based class label comparison*. For the ith ($1 \leq i \leq N$) example (x_i, y_i) in training set D_{tr}, it first finds the k nearest neighbors of (x_i, y_i) in D_{tr} using the Euclidean distance metric, and then compares the class label y_i of (x_i, y_i) with the most prevalent label y of these k neighbors: if $y_i \neq y$, then (x_i, y_i) is considered to be a noisy example, otherwise a non-noisy one. All the examples that are considered as noisy ones are directly removed from the original training set D_{tr}.
(4) ND_MulAda: a multi-class AdaBoost algorithm obtained by extending Cao et al.'s ND_AdaBoost [26] to the multi-class case. This extension is *not straightforward* but is achieved in the same way as our Rob_MulAda proposed in Section 4: first design a noise-detection based multi-class loss function as shown in subsection 4.2, then solve the minimization problem of this loss function by proving proposition one and thus get the optimal weight of each base classifier. The only difference between ND_MulAda and our Rob_MulAda lies in their adopted weight updating schemes: ND_MulAda adopts the one proposed in ND_AdaBoost (i.e., increases the weights

Table 2
Average test accuracies of the three algorithms under noise rates 5% and 10%.

Dataset	5%			10%		
	Ada	ND_Ada	ND_Ada*	Ada	ND_Ada	ND_Ada*
Australian	0.84709	0.85156	**0.86172**	0.82447	0.82548	**0.83361**
Breastcancer	0.94419	0.94516	**0.96018**	0.92581	0.93815	**0.94903**
Chess	0.94640	**0.95640**	0.95400	0.93360	0.92640	**0.93920**
Diabetes	0.72786	**0.73573**	0.731820	0.70746	0.71256	**0.72361**
German	**0.75825**	0.74694	0.74800	**0.73825**	0.72775	0.72200
Haberman	0.70931	0.71429	**0.73421**	0.67051	0.69728	**0.71674**
Heart	0.78093	0.78712	**0.79894**	0.74839	0.75558	**0.76997**
Indianliver	0.67329	0.67626	**0.69605**	0.66359	0.66928	**0.68205**
Ionosphere	0.89483	0.89488	**0.90612**	0.87889	**0.89379**	0.88857
Liverdisorder	0.70831	0.70255	**0.71987**	0.67353	0.67476	**0.68569**
Monks	0.98561	**0.99291**	0.99281	0.97482	0.97065	**0.98381**
Sonar	0.76059	0.76075	**0.77001**	0.75476	0.74508	**0.75897**
Splice	0.92750	0.92500	**0.92833**	0.89750	**0.90833**	0.90167
Tictactoe	0.92173	0.92482	**0.93520**	0.88450	0.89740	**0.90883**
Vertebral	0.81796	0.80809	**0.81804**	0.78527	0.78091	**0.79223**
Overall result	1-3-11	3-8-4	11-4-0	1-4-10	2-9-4	12-2-1
Average rank	2.67	2.07	1.27	2.60	2.13	1.27

Table 3
Average test accuracies of the three algorithms under noise rates 15% and 20%.

Dataset	15%			20%		
	Ada	ND_Ada	ND_Ada*	Ada	ND_Ada	ND_Ada*
Australian	0.79594	0.78725	**0.80931**	0.76124	0.75502	**0.77126**
Breastcancer	0.91094	0.90039	**0.92919**	0.88599	0.89329	**0.90616**
Chess	0.90080	0.89160	**0.91123**	0.86800	0.87360	**0.88670**
Diabetes	0.67376	**0.70605**	0.68985	0.65663	0.67321	**0.68630**
German	0.70525	0.71500	**0.71925**	0.68550	0.69175	**0.70853**
Haberman	0.64377	0.65862	**0.67051**	0.63027	0.63546	**0.65672**
Heart	0.71661	0.72621	**0.74691**	0.70253	0.71610	**0.72988**
Indianliver	0.66063	0.65583	**0.67430**	0.64484	0.65180	**0.66214**
Ionosphere	0.86898	0.87307	**0.88304**	0.83449	**0.85640**	0.84434
Liverdisorder	0.64633	0.65073	**0.66810**	0.62326	0.62629	**0.64348**
Monks	0.93165	**0.95568**	0.94694	0.87777	0.88313	**0.89589**
Sonar	0.71558	**0.73415**	0.73033	0.69352	0.69867	**0.71106**
Splice	0.87833	0.88145	**0.89417**	0.85583	**0.87167**	0.86925
Tictactoe	0.83036	0.84851	**0.86459**	0.81352	0.82576	**0.83996**
Vertebral	0.75250	0.76683	**0.78258**	0.74175	0.75519	**0.76932**
Overall result	0-4-11	3-8-4	12-3-0	0-1-14	2-12-1	13-2-0
Average rank	2.87	1.93	1.20	2.93	1.99	1.13

of misclassified non-noisy examples and correctly classified noisy examples) while Rob_MulAda adopts the one proposed in this paper.

5.2.3. *Experimental setting*

The experimental setting here is very similar to the one introduced in subsection 5.2.3 when comparing ND_AdaBoost and ND_AdaBoost* in two-class scenario. For the five compared algorithms, here we also adopt decision tree CART [34–36] as the base classifier due to that it is not only effective and unstable but also a multi-class classification algorithm. Similarly, all the experiments are carried out in the MatLab software with version 7.11, and the decision tree CART is implemented using the 'classregtree' function contained in the MatLab tool box. For each considered data set D in Table 5 and a given mislabeled noise rate, we also run five times four-fold cross validation to compare our algorithm Rob_MulAda with the other four algorithms mentioned above, and the size of ensembles generated by these algorithms is set to 25 except the single classifier algorithm SIN. The main steps of four-fold cross validation have already been given in Algorithm 5.

5.2.4. *Experimental results and analysis*

As in Section 5.1, we investigate the generalization performances of the five compared algorithms under varying mislabeled noise rates: 5%, 10%, 15%, 20%, 25% and 30%. For each considered multi-class data set D in Table 5 and a given noise rate, after five times four-fold cross validation is executed, the average test accuracies of the five algorithms are obtained: $\overline{accu}_{SIN}$, $\overline{accu}_{MulAda}$, $\overline{accu}_{NR\_MulAda}$, $\overline{accu}_{ND\_MulAda}$, $\overline{accu}_{Rob\_MulAda}$, which are used to compare their generalization performances in the mislabeled noisy training scenario. In the first part of this subsection, we present the generalization performances of the five algorithms under each of the six considered noise rates, and then in the second part, we depict the changes of their generalization performances with the increase of the noise rate for 6 data sets.

(1) *Generalization performances under different noise rates*

Tables 6–11 present the average test accuracies of the five compared algorithms under the six different noise rates, respectively. In these tables, 'NR_MAd', 'ND_MAd' and 'Rob_MAd' represent algorithms 'NR_MulAda', 'ND_MulAda' and 'Rob_MulAda' mentioned in subsection 5.2.2, respectively.

In Table 6, each row presents the average test accuracies of the five algorithms on a data set with mislabeled noise rate 5%, the highest accuracy in each row is marked in bold face while the second highest value is marked in italic. Similar with the tables in subsection 5.2.4, the last two rows in Table 6 summarize the comparison results of the five

Table 4
Average test accuracies of the three algorithms under noise rates 25% and 30%.

Dataset	25%			30%		
	Ada	ND_Ada	ND_Ada*	Ada	ND_Ada	ND_Ada*
Australian	0.73873	0.74829	**0.75493**	0.71952	0.72762	**0.73506**
Breastcancer	0.84123	0.85300	**0.87525**	0.79763	0.81056	**0.81315**
Chess	0.80520	**0.81520**	0.81120	0.76840	0.75920	**0.77520**
Diabetes	0.64773	0.65943	**0.67099**	0.63015	**0.63743**	0.63411
German	0.66750	0.67225	**0.68650**	**0.65025**	0.63850	0.63975
Haberman	0.60940	0.60155	**0.61113**	0.57979	0.58398	**0.60459**
Heart	0.68537	0.69620	**0.71197**	0.66173	0.66740	**0.67833**
Indianliver	0.62271	0.63318	**0.65109**	0.61310	0.61377	**0.62978**
Ionosphere	0.80078	**0.81770**	0.81652	0.75940	**0.76646**	0.75517
Liverdisorder	0.60022	0.60837	**0.62609**	0.56083	0.58401	**0.60288**
Monks	0.80932	0.81942	**0.82176**	0.76082	**0.78246**	0.77436
Sonar	**0.68330**	0.67942	0.67829	**0.67510**	0.65548	0.65462
Splice	**0.84583**	0.83330	0.84190	**0.81917**	0.80583	0.80617
Tictactoe	**0.77446**	0.77289	0.76976	**0.76659**	0.75919	0.75138
Vertebral	0.73499	0.74194	**0.75122**	**0.71814**	0.71609	0.71298
Overall result	3-1-11	2-11-2	10-3-2	5-2-8	3-9-3	7-4-4
Average rank	2.53	2.00	1.47	2.20	2.00	1.80

Table 5
Characteristics of the used multi-class data sets.

No.	Data set	Examples	Attributes	Classes
1	Balancescale	625	4	3
2	Contraceptive	1473	9	3
3	Ecoli	336	7	8
4	Iris	150	4	3
5	Led7digit	500	7	10
6	Newthyroid	215	5	3
7	Pendigits	7494	16	10
8	Satimage	4435	36	6
9	Seeds	210	7	3
10	Segmentation	2310	19	7
11	Shuttle	43,500	9	7
12	Svmguide2	391	20	3
13	Svmguide4	612	10	6
14	Userknowledge	403	5	4
15	Vehicle	846	18	4
16	Vowel	990	10	11
17	Wine	178	13	3
18	Yeast	1484	8	10

algorithms over all the 18 data sets. The 'Overall result' row presents the numbers of data sets on which each algorithm achieves the highest average test accuracy, the second highest accuracy, ..., the lowest accuracy, respectively. For instance, the numbers '4-3-9-2-0' in the second column represent that the multi-class AdaBoost algorithm MulAda achieves the highest average test accuracy on 4 out of the 18 data sets, the second and the third highest accuracies on 3 and 9 data sets respectively, while the second lowest and the lowest accuracies on 2 and 0 data sets respectively. The 'Average rank' row gives the average rank of each algorithm that is obtained by averaging its ranks over all the 18 data sets. The remaining tables in the following can be understood in the same way.

When the mislabeled noise rate is 5%, it can be seen from Table 6 that all the four boosting algorithms can significantly improve the generalization performance of the single classifier SIN; the two algorithms Rob_MulAda and ND_MulAda can improve the generalization performance of the multi-class AdaBoost algorithm MulAda in most cases, but the improvement is not obvious and they have similar test accuracies as MulAda, i.e., MulAda is not obviously affected by this low noise rate; the algorithm NR_MulAda improves the generalization performance of MulAda only in 6 data sets and in most cases has lower test accuracy than MulAda. Generally, NR_MulAda is not effective under this noise rate, which can be reflected by its high average rank 3.17.

When the mislabeled noise rate is 10%, it can be seen from Table 7 that the proposed algorithm Rob_MulAda achieves the highest average test accuracy on 9 data sets, the second and the third highest accuracies on 6 and 3 data sets, respectively. In addition, Rob_MulAda also has the lowest average rank 1.67 among the compared algorithms; ND_MulAda achieves the highest and the second highest accuracies on 3 and 8 data sets respectively, while NR_MulAda gets the highest and the second highest accuracies on 5 and 3 data sets respectively. Generally, Rob_MulAda can significantly improve the generalization performance of MulAda and has the best generalization performance among the compared algorithms, and besides, the relative performance of NR_MulAda improves and it is nearly comparable with ND_MulAda in many cases.

When the mislabeled noise rate is moderate, i.e., $noise\_rate \in \{15\%, 20\%\}$, we can learn from Tables 8 to 9 that, our algorithm Rob_MulAda achieves the highest or second highest average test accuracies on about 15 data sets and has the lowest average rank. Thus, it has the best overall generalization performance among the compared algorithms; the relative performance of NR_MulAda improves with the increase of the noise rate and it is almost comparable with ND_MulAda in most cases, the two algorithms both achieve the highest or second highest average test accuracies on roughly 10 data sets, and besides, their average ranks are very approximate. On the whole, the above three algorithms can well improve the generalization performance of the multi-class AdaBoost algorithm MulAda but with Rob_MulAda being the most effective one.

When the mislabeled noise rate is high, i.e., $noise\_rate \in \{25\%, 30\%\}$, it can be learned from Tables 10 to 11 that, different from the cases of low and moderate noise rates, NR_MulAda gets the highest average test accuracy on about 12 data sets and has the lowest average rank, thus becomes the best in overall generalization performance; Rob_MulAda gets the highest or second highest average test accuracies on about 12 data sets and ND_MulAda on about 10 data sets. In most cases, the generalization performance of Rob_MulAda is better than or competitive with that of ND_MulAda, which is consistent with their relation in the cases of low and

Table 6
Average test accuracies of the five algorithms under noise rate 5%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.78049	0.82178	*0.82370*	0.82273	**0.83106**
Contraceptive	0.48364	0.51962	**0.54704**	0.52165	*0.52532*
Ecoli	0.77440	0.81964	**0.82440**	0.81964	*0.82083*
Iris	0.92651	*0.94671*	**0.95218**	*0.94671*	0.94272
Led7digit	0.68560	0.70440	0.69920	*0.71120*	**0.71200**
Newthyroid	0.91611	0.92906	0.91243	**0.93383**	*0.93273*
Pendigits	0.83233	**0.91800**	0.90833	0.91200	*0.91788*
Satimage	0.76967	0.84500	0.82933	**0.84833**	*0.84567*
Seed	0.89420	0.90178	0.88095	*0.90381*	**0.92851**
Segment	0.89200	*0.93175*	0.92425	0.93125	**0.94575**
Shuttle	0.98367	*0.99467*	**0.99800**	0.99383	0.99433
Svmguide2	0.62549	**0.74422**	0.72416	*0.73706*	0.73445
Svmguide4	0.74000	**0.79067**	0.77267	0.78067	*0.78333*
Userknowledge	0.89383	0.91394	0.90842	*0.91452*	**0.92194**
Vehicle	0.68773	**0.73524**	0.70380	*0.73169*	0.72849
Vowel	0.72917	*0.84242*	0.72992	0.84129	**0.84318**
Wine	0.89886	0.94605	0.90117	*0.94615*	**0.95178**
Yeast	0.52628	0.57547	**0.58531**	*0.57709*	0.57466
Overall result	0-0-0-2-16	4-3-9-2-0	5-1-0-10-2	2-8-6-2-0	7-6-3-2-0
Average rank	4.89	2.50	3.17	2.44	2.00

Table 7
Average test accuracies of the five algorithms under noise rate 10%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.77086	0.81407	*0.81916*	**0.82235**	0.81696
Contraceptive	0.47172	0.51011	**0.53536**	*0.51690*	0.51378
Ecoli	0.75774	0.80200	**0.82155**	0.80725	*0.81500*
Iris	0.91292	0.92231	**0.94055**	0.92772	*0.92900*
Led7digit	0.67760	0.70000	0.68824	**0.70220**	*0.70040*
Newthyroid	0.88438	0.91401	0.90901	*0.92000*	**0.93095**
Pendigits	0.79720	0.89400	*0.90580*	0.90280	**0.91680**
Satimage	0.74267	0.82867	0.81000	**0.83300**	*0.82933*
Seed	0.85752	0.87586	0.87871	*0.88957*	**0.90648**
Segment	0.86875	0.91825	0.91950	*0.92225*	**0.93775**
Shuttle	0.95317	0.98700	**0.99800**	*0.99050*	0.98867
Svmguide2	0.61729	0.70008	0.71875	*0.72213*	**0.73219**
Svmguide4	0.69867	**0.76933**	0.75733	0.76267	*0.76533*
Userknowledge	0.86111	0.89179	*0.90123*	0.89762	**0.91123**
Vehicle	0.64605	0.70910	0.69808	*0.71232*	**0.72334**
Vowel	0.69167	*0.81818*	0.70152	0.80947	**0.82856**
Wine	0.85499	0.90863	0.89661	*0.91670*	**0.93011**
Yeast	0.50323	0.56779	**0.58450**	0.56954	*0.57143*
Overall result	0-0-0-0-18	1-1-5-11-0	5-3-3-7-0	3-8-7-0-0	9-6-3-0-0
Average rank	5.00	3.44	2.67	2.22	1.67

Table 8
Average test accuracies of the five algorithms under noise rate 15%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.76446	*0.80990*	0.79933	0.80444	**0.81439**
Contraceptive	0.46107	0.50025	**0.52517**	*0.51255*	0.51119
Ecoli	0.73571	0.79085	**0.81920**	0.79286	*0.80119*
Iris	0.86186	0.89861	**0.93076**	0.90996	*0.91807*
Led7digit	0.66520	0.69120	0.67640	**0.69880**	*0.69240*
Newthyroid	0.83713	0.90000	*0.90499*	*0.90499*	**0.91797**
Pendigits	0.78440	0.88720	0.89200	**0.90000**	*0.89940*
Satimage	0.70567	0.79300	0.80033	*0.81633*	**0.82033**
Seed	0.84291	0.86145	0.87019	*0.87486*	**0.89215**
Segment	0.82400	0.89650	**0.91950**	0.90100	*0.91400*
Shuttle	0.92550	0.97383	**0.99800**	0.98100	*0.98333*
Svmguide2	0.60203	0.69077	*0.71347*	0.71005	**0.71993**
Svmguide4	0.66067	0.74067	0.72600	*0.74333*	**0.75400**
Userknowledge	0.82334	0.88773	*0.89821*	0.89073	**0.90271**
Vehicle	0.63002	0.70134	0.69486	*0.70608*	**0.72000**
Vowel	0.65947	**0.80303**	0.67765	*0.79053*	0.79015
Wine	0.80578	0.87230	0.85795	*0.90222*	**0.91334**
Yeast	0.48881	0.55970	**0.58396**	*0.56873*	0.56679
Overall result	0-0-0-0-18	1-1-4-12-0	6-3-3-6-0	2-8-8-0-0	9-6-3-0-0
Average rank	5.00	3.50	2.50	2.31	1.67

Table 9
Average test accuracies of the five algorithms under noise rate 20%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.74113	0.79008	*0.79323*	0.79264	**0.80500**
Contraceptive	0.45353	0.49465	**0.51143**	*0.49573*	0.49559
Ecoli	0.69643	0.77119	**0.80952**	0.78501	*0.79833*
Iris	0.84567	*0.88060*	**0.92748**	0.88032	0.87086
Led7digit	0.65200	!‘!‘0.68000	0.67200	*0.69040*	**0.69140**
Newthyroid	0.80167	0.87208	0.88417	*0.88864*	**0.90130**
Pendigits	0.74380	0.87000	**0.88920**	0.88180	*0.88502*
Satimage	0.66200	0.77467	0.79433	*0.80430*	**0.81767**
Seed	0.81419	0.84373	0.85315	*0.86139*	**0.88134**
Segment	0.79250	0.86500	**0.91050**	0.87950	*0.89025*
Shuttle	0.88300	0.96133	**0.99750**	0.96600	*0.97167*
Svmguide2	0.58252	0.66749	*0.68768*	0.68563	**0.70990**
Svmguide4	0.63467	*0.67667*	0.64933	*0.67667*	**0.69333**
Userknowledge	0.78751	0.85329	0.86543	*0.87829*	**0.89483**
Vehicle	0.60027	0.67019	0.68227	*0.69126*	**0.71214**
Vowel	0.63523	0.76780	0.66682	**0.78258**	*0.77235*
Wine	0.79770	0.85788	0.83170	**0.89563**	*0.89036*
Yeast	0.46685	0.53879	**0.57749**	*0.55458*	0.54704
Overall result	0-0-0-0-18	0-2-3-13-0	7-2-5-4-0	2-8-8-0-0	9-6-2-1-0
Average rank	5.00	3.64	2.33	2.31	1.72

Table 10
Average test accuracies of the five algorithms under noise rate 25%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.70281	0.75238	**0.77962**	*0.76363*	0.76300
Contraceptive	0.43534	0.47088	**0.50139**	0.47709	*0.48540*
Ecoli	0.67560	0.75753	**0.79879**	0.77571	*0.78943*
Iris	0.77616	0.83831	**0.89012**	0.85655	*0.86881*
Led7digit	0.64550	0.67600	0.65900	*0.68300*	**0.69000**
Newthyroid	0.77302	0.86161	0.87125	*0.87369*	**0.88458**
Pendigits	0.71275	0.86675	**0.88125**	*0.87000*	0.86875
Satimage	0.63225	0.76650	*0.78950*	0.78575	**0.80400**
Seed	0.78023	0.82141	0.84626	*0.85596*	**0.86600**
Segment	0.73313	0.86125	**0.89281**	0.86688	*0.87031*
Shuttle	0.84917	0.94467	**0.97550**	0.95100	*0.95533*
Svmguide2	0.53646	0.65034	*0.66205*	**0.67609**	0.66184
Svmguide4	0.60833	0.66783	0.63833	*0.67017*	**0.68483**
Userknowledge	0.75083	0.83945	**0.85376**	*0.84947*	0.84559
Vehicle	0.56440	0.65579	**0.67839**	*0.66819*	0.66731
Vowel	0.60180	*0.75336*	0.65294	**0.77036**	0.74953
Wine	0.77551	0.83449	0.81808	*0.84134*	**0.84566**
Yeast	0.45013	0.53313	**0.56716**	0.54431	*0.55206*
Overall result	0-0-0-0-18	0-1-3-14-0	10-2-2-4-0	2-9-7-0-0	6-6-6-0-0
Average rank	5.00	3.72	2.00	2.28	2.00

Table 11
Average test accuracies of the five algorithms under noise rate 30%.

Dataset	SIN	MulAda	NR_MAd	ND_MAd	Rob_MAd
Balancescale	0.68600	0.74360	**0.76438**	0.74518	*0.75160*
Contraceptive	0.42397	0.46436	**0.49476**	*0.46852*	0.46166
Ecoli	0.63467	0.74595	**0.78985**	0.75446	*0.76190*
Iris	0.76165	0.80808	**0.87775**	0.83039	*0.83901*
Led7digit	0.63300	*0.67100*	0.60750	0.66350	**0.68050**
Newthyroid	0.75388	0.84969	0.85988	*0.86591*	**0.87155**
Pendigits	0.67750	0.85125	**0.86800**	0.85400	*0.85475*
Satimage	0.60525	0.75600	**0.77925**	0.77025	*0.77150*
Seed	0.75200	0.81427	**0.84084**	0.83569	*0.84003*
Segment	0.68500	0.84125	**0.87281**	*0.84937*	0.84875
Shuttle	0.77983	0.92150	**0.95267**	*0.93033*	0.92950
Svmguide2	0.52609	0.63299	*0.64524*	**0.65952**	0.64402
Svmguide4	0.58667	0.65250	0.62400	*0.66500*	**0.67917**
Userknowledge	0.73025	0.81368	**0.84288**	0.81342	*0.82583*
Vehicle	0.52830	0.64330	**0.66718**	0.65689	*0.65919*
Vowel	0.59754	*0.74290*	0.62311	**0.74763**	0.73201
Wine	0.70689	0.81443	0.80428	*0.82856*	**0.83518**
Yeast	0.41560	0.52544	**0.55199**	*0.53454*	0.53251
Overall result	0-0-0-0-18	0-2-3-13-0	12-1-1-4-0	2-7-8-1-0	4-8-6-0-0
Average rank	5.00	3.61	1.83	2.44	2.11

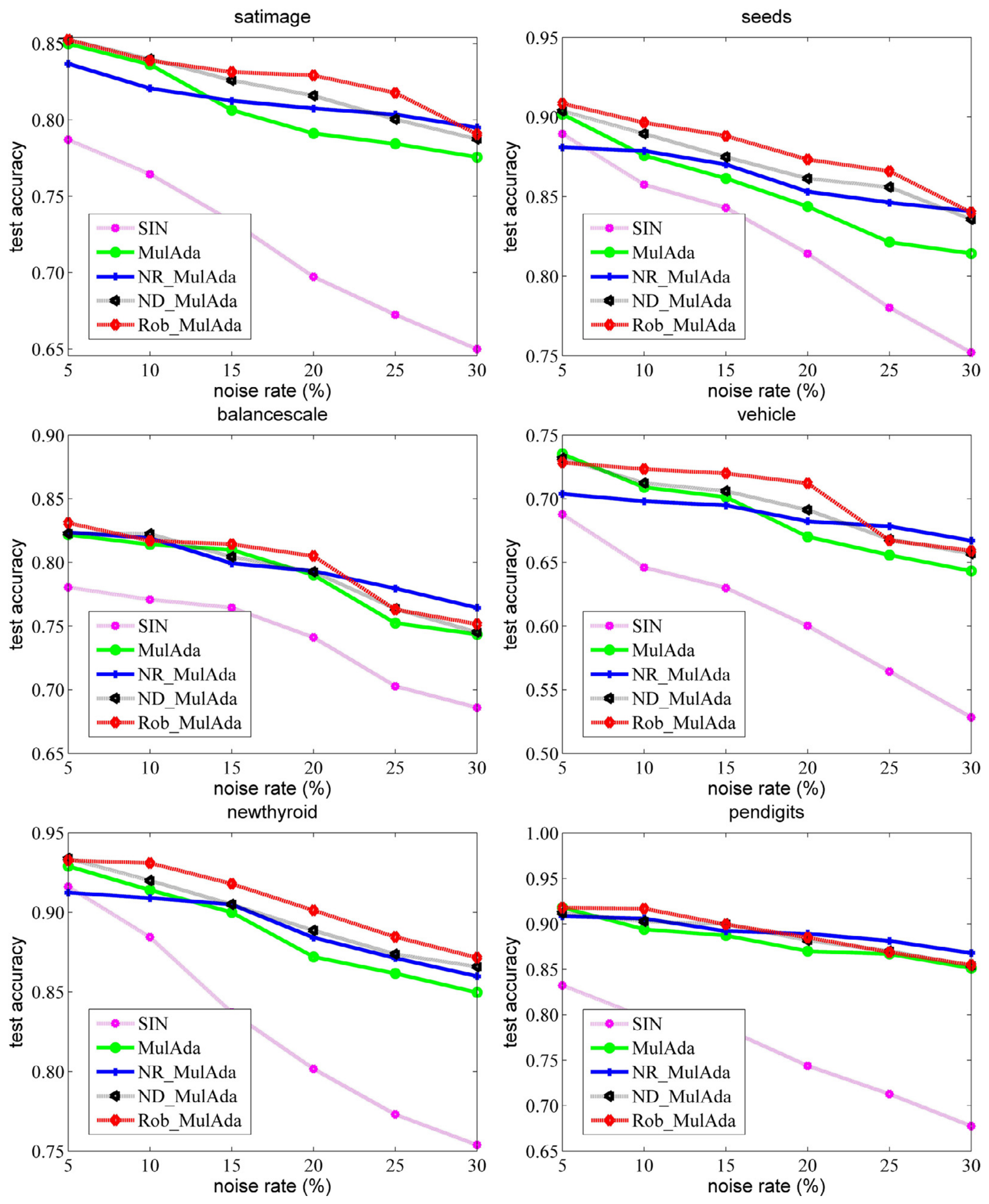


Fig. 2. The changes of average test accuracy with respect to noise rate.

moderate noise rates. Generally, in this high noise rate scenario, the above three algorithms can still improve the generalization performance of the multi-class AdaBoost algorithm MulAda while with NR_MulAda being the most effective one.

(2) *Changes of generalization performance with respect to noise rate*

For the five compared algorithms, to more clearly illustrate the changing process of their generalization performances with respect to the mislabeled noise rate, we depict the changes of their average test accuracies with the increase of the noise rate for 6 representative data sets (i.e., satimage, seeds, balancescale, vehicle, newthyroid and pendigits), which are shown in Fig. 2.

We can learn from Fig. 2 that, for each of the five algorithms and a given data set, it achieves its highest average test accuracy at the lowest noise rate 5%, and then its average test accuracy gradually decreases with the increase of the noise rate; when the noise rate is low, i.e., 5% and 10%, the proposed algorithm Rob_MulAda generally has the best generalization performance, followed by the algorithm ND_MulAda, while NR_MulAda can not improve the generalization performance of the multi-class AdaBoost algorithm MulAda in many cases; when the noise rate is moderate, i.e., 15% and 20%, Rob_MulAda remains has the best overall generalization performance, and NR_MulAda gets a comparable performance with ND_MulAda in many cases, i.e., the relative performance of NR_MulAda improves; when the noise rate is high, i.e., 25% and 30%, NR_MulAda outperforms Rob_MulAda and ND_MulAda in most cases and achieves the best overall generalization performance, followed by Rob_MulAda which is better than or competitive with ND_MulAda in most cases. The three algorithms still can improve the generalization performance of MulAda in this high noise rate scenario, while the four boosting algorithms can significantly improve the generalization performance of the single classifier SIN under all the considered noise rates.

5.2.5. Summary of experimental results

The above obtained experimental results are summarized as follows:

When the mislabeled noise rate is low ($noise\_rate \in (0, 10\%]$) and moderate ($noise\_rate \in (10\%, 20\%]$), in general, the proposed algorithm Rob_MulAda can most effectively improve the generalization performance of the multi-class AdaBoost algorithm MulAda and has the lowest average rank. Therefore, Rob_MulAda outperforms the other compared algorithms; Generally, the approach ND_MulAda has an inferior generalization performance to Rob_MulAda, indicating that our weight updating scheme is also more effective in the multi-class classification scenario. NR_MulAda can not effectively improve the generalization performance of MulAda when the noise rate is low, however, its relative performance improves for the moderate noise rates while it is competitive with ND_MulAda in most cases. These observations indicate that, *when the noise rate is low or moderate, our noise-detection based weight adjustment approach Rob_MulAda is most effective in alleviating the harmful effect of mislabeled noises, while the noise-removal based approach NR_MulAda shrinks the training set size and is not so effective.*

When the noise rate is high ($noise\_rate \in (20\%, 30\%]$), both NR_MulAda and Rob_MulAda can well improve the generalization performance of MulAda with NR_MulAda being more effective than Rob_MulAda, while ND_MulAda is not very effective in many cases. On the whole, NR_MulAda outperforms the proposed algorithm Rob_MulAda and ND_MulAda and becomes the best algorithm, indicating that *in the training scenario with high-level noises, our noise-detection based weight adjustment approach Rob_MulAda is no longer the most effective and instead directly removing those noisy examples could be a better option.*

To sum up, we expect that the above finding could serve as a suggestion for choosing the most appropriate approach according to the concrete noise level in practical applications to effectively enhance the robustness of multi-class AdaBoost.

6. Conclusion and future work

AdaBoost is a very successful and popular ensemble learning algorithm, however, it is sensitive to mislabeled noisy examples and non-robust to them. Most existing works focus on improvement of the robustness of the classical two-class AdaBoost algorithm, unfortunately, their proposed approaches are hard to be applied to the multi-class classification scenario. Inspired by the approach ND_AdaBoost in the two-class classification scenario, in this paper, we propose a robust multi-class AdaBoost algorithm Rob_MulAda, in which a noise-detection based multi-class loss function is formally designed and a new weight updating scheme is presented to mitigate the harmful effect of noisy examples.

To verify the robustness of our weight updating scheme, we first directly compare ND_AdaBoost with its modified version ND_AdaBoost* in the two-class classification scenario, where ND_AdaBoost* is purposely designed by just substituting the weight updating scheme of ND_AdaBoost with ours. The conducted experiments on 15 two-class benchmark data sets indicate that our weight updating scheme is indeed more robust to mislabeled noises than that of ND_AdaBoost in the two-class classification scenario. Then, we compare the proposed Rob_MulAda with other four closely-related algorithms under varying mislabeled noise rates by carrying out experiments on 18 multi-class benchmark data sets. Experimental results demonstrate that when the noise rate is low or moderate, our Rob_MulAda has the best overall generalization performance, followed by ND_MulAda, while the noise-removal based approach NR_MulAda is not so effective in many cases; when the noise rate is high, the noise-removal based approach NR_MulAda becomes the best, while Rob_MulAda is no longer the most effective but is still better than or competitive with ND_MulAda in most cases. These results indicate that our weight updating scheme is also more robust than that of ND_AdaBoost in the multi-class scenario. Based on the above observations we suggest that, in practical applications, *when the mislabeled noise rate is low or moderate, it is better to use the proposed Rob_MulAda to alleviate the harmful influence of mislabeled noises, and when the noise rate is high, directly removing the noisy examples could be the most appropriate option.*

As our Rob_MulAda belongs to the family of sequential algorithms, it can not generate base classifiers independently and simultaneously like the parallel ensemble learning algorithm Bagging. Thus, Rob_MulAda achieves low efficiency especially when applied to big data with a huge amount of training examples. As far as we know, there does not exist any research on the improvement of the efficiency of the multi-class AdaBoost algorithms. Therefore, improving the efficiency of the multi-class AdaBoost algorithms including our Rob_MulAda using a certain technique (e.g., distributed learning) is an interesting work that deserves further study.

Acknowledgement

We would like to express our gratitude to both the editor and the anonymous reviewers for their constructive comments that improved the quality of our manuscript to a large extent. This work is supported by the National Natural Science Foundation of China (Nos. 61501229, 61472186), the Fundamental Research Funds for the Central Universities (Nos. NS2015091, NJ20160013), and the Jiangsu Qinglan Project.

References

[1] Z.H. Zhou, Ensemble methods: Foundations and Algorithms, CRC Press, Florida, 2012.
[2] L. Rokach, Ensemble-based classifiers, Artif. Intell. Rev. 33 (1–2) (2010) 1–39.
[3] A. Rahman, B. Verma, Ensemble classifier generation using non-uniform layered clustering and genetic algorithm, Knowl.Based Syst. 43 (1) (2013) 30–42.
[4] G. Martïnez-Munoz, A. Suärez, Switching class labels to generate classification ensembles, Pattern Recog. 38 (10) (2005) 1483–1494.
[5] R.E. Banfield, L.O. Hall, KW. Bowyer, A comparison of decision tree ensemble creation techniques, IEEE Trans. Pattern Anal. Mach. Intell. 29 (1) (2007) 173–180.
[6] J. Lu, K.N. Plataniotis, A.N. Venetsanopoulos, Ensemble-based discriminant learning with boosting for face recognition, IEEE Trans. Neural Netw. 17 (1) (2006) 166–178.
[7] M. Galar, A. Fernandez, E. Barrenechea, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst. Man Cybern. 42 (4) (2012) 463–484.
[8] L.L. Minku, X. Yao, DDD: a new ensemble approach for dealing with concept drift, IEEE Trans. Knowl. Data Eng. 24 (4) (2012) 619–633.
[9] N. Lu, G. Zhang, J. Lu, Concept drift detection via competence models, Artif. Intell. 209 (1) (2014) 11–28.
[10] L. Breiman, B. predictors, Mach. Learn. 24 (2) (1996) 123–140.
[11] L. Breiman, R. forests, Mach. Learn. 45 (1) (2001) 5–32.
[12] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization, Mach. Learn. 40 (2) (2000) 139–157.
[13] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1) (1997) 119–139.
[14] L.V. Utkin, Y.A. Zhuk, Robust boosting classification models with local sets of probability distributions, Knowl. Based Syst. 61 (1) (2014) 59–75.
[15] I. Landesa-Vazquez, J.L. Alba-Castro, Shedding light on the asymmetric learning capability of adaboost, Pattern Recog. Lett. 33 (3) (2012) 247–255.
[16] P. Melville, R.J. Mooney, Creating diversity in ensembles using artificial data, Inf. Fusion 6 (1) (2005) 99–111.
[17] B. Sun, H.Y. Chen, J.D. Wang, An empirical margin explanation for the effectiveness of DECORATE ensemble learning algorithm, Knowl. Based Syst. 78 (1) (2015) 1–12.
[18] F. Muhlenbach, S. Lallich, D.A. Zighed, Identifying and handling mislabeled instances, J. Intell. Inf. Syst. 22 (1) (2004) 89–109.
[19] M. Galar, A. Fernändez, E. Barrenechea, An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes, Pattern Recog. 44 (8) (2011) 1761–1776.
[20] J.A. Säez, M. Galar, J. Luengo, Analyzing the presence of noise in multi-class problems: alleviating its influence with the one-vs-one decomposition, Knowl. Inf. Syst. 38 (1) (2014) 179–206.

[21] J. Zhu, H. Zou, S. Rosset, T. Hastie, Multi-class AdaBoost, Stat. Its Interface 2 (1) (2009) 349–360.

[22] A. Angelova, Y. Abu-Mostafam, P. Perona, Pruning training sets for learning of object categories, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, Los Alamitos, 2005, pp. 494–501.

[23] C. Domingo, O. Watanabe, MadaBoost: a modification of AdaBoost, Proceedings of the Computational Learning Theory, Tokyo Industry Technology, Tokyo, 2000, pp. 180–189.

[24] R.A. Servedio, Smooth boosting and learning with malicious noise, J. Mach. Learn. Res. 4 (4) (2003) 633–648.

[25] Y. Gao, F. Gao, Edited adaboost by weighted kNN, Neurocomputing 73 (16) (2010) 3079–3088.

[26] J. Cao, S. Kwong, R. Wang, A noise-detection based AdaBoost algorithm for mislabeled data, Pattern Recog. 45 (1) (2012) 4451–4465.

[27] I. Mukherjee, R.E. Schapire, A theory of multi-class boosting, Advances in Neural Information Processing Systems, The MIT Press, Cambridge, 2010, pp. 1–9.

[28] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, Proceedings of the 13th International Conference on Machine Learning, Morgan Kaufmann Publishers, San Francisco, 1996, pp. 148–156.

[29] R.E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Mach. Learn. 37 (3) (1999) 297–336.

[30] R. Schapire, Y. Singer, BoosTexter: a boosting-based system for text categorization, Mach. Learn. 39 (2) (2000) 135–168.

[31] X. Jin, X. Hou, C.L. Liu, Multi-class AdaBoost with hypothesis margin, Proceedings of the 20th International Conference on Pattern Recognition, IEEE Computer Society, Los Alamitos, 2010, pp. 65–68.

[32] A. Asuncion, D. Newman, UCI machine learning repository. http://www.ics.uci.edu/~mlearn/mlrepository.html.

[33] C. Chang, C. Lin, Libsvm: a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

[34] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Tree, Chapman & Hall, New York, 1984.

[35] S.C. Lemon, J. Roy, M.A. Clark, Classification and regression tree analysis in public health: methodological review and comparison with logistic regression, Ann. Behavior. Med. 26 (3) (2003) 172–181.

[36] W.Y. Loh, Classification and regression trees, Data Mining Knowl. Discov. 1 (1) (2011) 14–23.