Boosting for Straggling and Flipping Classifiers

Yuval Cassuto*, Yongjune Kim†
*Viterbi Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel
†Department of Information and Communication Engineering, DGIST, Daegu 42988, South Korea
Email: *ycassuto@ee.technion.ac.il, yjk@dgist.ac.kr*

***Abstract*—Boosting is a well-known method in machine learning for combining multiple weak classifiers into one strong classifier. When used in distributed setting, accuracy is hurt by classifiers that flip or straggle due to communication and/or computation unreliability. While unreliability in the form of noisy data is well-treated by the boosting literature, the unreliability of the classifier outputs has not been explicitly addressed. Protecting the classifier outputs with an error/erasure-correcting code requires reliable encoding of multiple classifier outputs, which is not feasible in common distributed settings. In this paper we address the problem of training boosted classifiers subject to straggling or flips at classification time. We propose two approaches: one based on minimizing the usual exponential loss but in expectation over the classifier errors, and one by defining and minimizing a new worst-case loss for a specified bound on the number of unreliable classifiers.**

I. Introduction

To enable inference tasks on computationally limited devices, *ensemble methods in machine learning* [1] were devised to combine multiple weak learners to one powerful learner. The ensemble paradigm is especially suitable for distributed inference, in which multiple weak nodes join together to perform an ambitious inference task. When deployed in a distributed setting, the nodes are subject to *unreliability*, manifested in instances where they fail to correctly deliver their output toward the combined result. In this paper we specifically address the *binary classification* inference task, and consider unreliability in the form of nodes that flip their value, or fail (straggle) to give any value.

Ensemble classification schemes train an ensemble of (weak) *base classifiers*, and specify how the final classification output is aggregated from the base-classifier outputs. The piece missing in the art of ensemble methods for distributed classification is *how to deal with the unreliability of the nodes realizing the base classifiers*. This extrinsic unreliability is distinct from the intrinsic inaccuracy of the base classifiers, because it feeds the aggregate classification with inputs that differ from what the base classifiers were designed and trained to output. Prior work on ensemble reliability mostly focuses on noise in the data labels (see e.g. [2] and survey therein), or assumes that training and classification are performed on the same unreliable hardware [3], [4]. Recently, [5] addressed the problem of resource allocation for unreliable ensembles, assuming the classifiers are given post-training.

The particular ensemble method we pursue in this paper for distributed classification is *adaptive boosting (Adaboost)* [6], owing to its generality, its proven optimization properties, and its performance superiority in many classification tasks. The main feature of Adaboost (and arguably its source of advantage) is that the base classifiers are trained adaptively in sequence, whereby each classifier is trained with a weighting of the training data points that is adapted according to the accuracy of previous classifiers in the sequence. Our main contribution in this paper is toward extending the Adaboost scheme to handle base classifiers that are unreliable in two natural ways: 1) classifiers that *straggle* and thus their output is unavailable to the aggregator, and 2) classifiers that deliver a *flipped* binary value relative to the designed-classifier's output. It turns out that erasure/error correcting codes that look natural for dealing with straggling/flipping are not ideal for the distributed ensemble setup, because they require reliable encoding of multiple classifier outputs. Instead, our approach is to consider the unreliability model (straggling or flipping) in the training of the ensemble, such that the aggregator can achieve better classification accuracy (success over the test set) when combining these classifiers.

The first part of the paper (Sections III, IV) presents boosting algorithms for a *specified probability* of straggling (Section III) and flipping (Section IV). Similarly to the ordinary Adaboost algorithm, the proposed algorithms guarantee optimal exponential loss by each classifier in the sequence, but do so in expectation over the possible subsets of straggling/flipping classifiers. We show the performance improvements of these algorithms over a synthetic dataset, and also compare to the non-adaptive ensemble method called *bagging* (bootstrap aggregation) [7]. In the second part (Section V), we depart from the i.i.d. probabilistic model to a worst-case model of mitigating any *single flipping* classifier in the ensemble. For this setting we introduce a new loss measure we call the ensemble's *1-flip loss*, which captures the classification sensitivity to the worst flipping classifier in the ensemble. To lower the 1-flip loss, we propose a boosting algorithm we call *reliability-regularized Adaboost*, and suggest an efficient bisection algorithm to look for the regularization parameter that minimizes the 1-flip loss. Evaluating on the widely-available CBCL face-detection dataset [8], the new scheme is shown to give better 1-flip accuracies compared to ordinary Adaboost.

∗ Work supported in part by the US-Israel Binational Science Foundation and by the Israel Science Foundation.

† Work supported in part by the Daegu Gyeongbuk Institute of Science and Technology (DGIST) Start-up Fund Program of the Ministry of Science and ICT under Grant 2021010014.

II. Review of the Adaboost Algorithm

Consider a training dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ where each x_i is a *data vector* and each y_i is a *binary label* in $\{-1, 1\}$; a pair (x_i, y_i) is called a *data point*. The elementary building block of the Adaboost classifier is a *base classifier* $h(\cdot)$ mapping x to a binary label $y \in \{-1, 1\}$; h is trained to minimize an *empirical risk* $R_{\mathcal{D},W}(h) = \sum_{i=1}^{m} w_i I(h(x_i) \neq y_i)$, in which data points are weighted by $W = (w_i)_{i=1}^{m} : \sum_{i=1}^{m} w_i = 1$, and $I(e)$ is the indicator function that evaluates to 1 when event e occurs and to 0 otherwise. The Adaboost algorithm uses $\mathcal{D}$ to train a *combined classifier* of the form $H(x) = \text{sign}\left(\sum_{t=1}^{M} \alpha_t h_t(x)\right)$, where each $h_t(\cdot)$, $t \in \{1, \ldots, M\}$, is a base classifier. A well known result on the classifiers h_t and their coefficients α_t output by the Adaboost algorithm is the following.

Proposition 1 *([9]) At each iteration t of Adaboost, the classifier h_t and the classifier coefficient α_t minimize the exponential loss $\sum_{i=1}^{m} \exp\left\{-y_i \sum_{j=1}^{t} \alpha_j h_j(x_i)\right\}$, given the classifiers $h_1, \ldots, h_{t-1}$ and their coefficients $\alpha_1, \ldots, \alpha_{t-1}$ found in previous iterations.*

III. Boosting with Straggling Classifiers

In a distributed-classification setup, at classification time an aggregator node may face the issue of *straggling* base classifiers failing to deliver their input for the combined classifier. In the *i.i.d. straggling* model, each classifier straggles with probability η, independently from other classifiers. Formally, now $\tilde{H}(x) = \text{sign}\left(\sum_{t=1}^{M} \alpha_t \tilde{h}_t(x)\right)$, where $\tilde{h}_t(x) = h_t(x)$ with probability $1 - \eta$ and $\tilde{h}_t(x) = 0$ with probability η. To mitigate the potential degradation of performance, we next modify the Adaboost training procedure to address straggling classifiers. Note that at training time the identities of the straggling classifiers are not known (in fact, these may change between classification instances), and we only assume the straggling probability η is known. For a set S of base classifiers, a subset $A \subseteq S$ defines a partition of S to *non*-straggling classifiers in A and straggling classifiers in $S \setminus A$. We denote by $P_\eta(A)$ the probability that $(A, S \setminus A)$ is the partition of S to non-straggling,straggling classifiers. Let $|X|$ denote the size of a set X; for the i.i.d. straggling model we have

$$P_\eta(A) = (1 - \eta)^{|A|} \cdot \eta^{|S|-|A|}. \tag{1}$$

A. Adaboost training for straggling classifiers

Instead of just ignoring straggling classifiers (by taking their values as 0s), a better approach is to train the individual classifiers $h_t(\cdot)$ *taking into account that they straggle* i.i.d. with probability η. Recall from Proposition 1 that Adaboost training performs sequential minimization of an exponential loss function, that is, looking for h_t, α_t that minimize

$$E = \sum_{i=1}^{m} \exp\{-y_i S_t(x_i)\} = \sum_{i=1}^{m} \prod_{j=1}^{t} \exp\{-y_i \alpha_j h_j(x_i)\}, \tag{2}$$

where $S_t(x) \triangleq \sum_{j=1}^{t} \alpha_j h_j(x)$, and we know $\{h_j, \alpha_j\}_{j=1}^{t-1}$ from previous iterations of the algorithm. In the ordinary case (with no stragglers) we can now rewrite

$$E = \sum_{i=1}^{m} \exp\{-y_i S_{t-1}(x_i)\} \exp\{-y_i \alpha_t h_t(x_i)\}, \tag{3}$$

and minimize only the second exponent. With stragglers, however, we replace $S_t(\cdot)$ by $\tilde{S}_t(x) \triangleq \sum_{j=1}^{t} \alpha_j \tilde{h}_j(x)$ in (2), and the latter is not deterministically known at training time. So instead we need to minimize the *expected* exponential loss function

$$\bar{E} = \sum_{i=1}^{m} \sum_{A \subseteq [1,t]} P_\eta(A) \prod_{\substack{j=1 \\ j \in A}}^{t} \exp\{-y_i \alpha_j h_j(x_i)\}, \tag{4}$$

where the expectation is taken as a sum over all subsets $A \subseteq [1, t]$ of non-straggling classifiers. From the i.i.d. decomposition of $P_\eta(A)$ in (1) and using the distributive law we get

$$\bar{E} = \sum_{i=1}^{m} \prod_{j=1}^{t} \left[(1 - \eta) \exp\{-y_i \alpha_j h_j(x_i)\} + \eta\right]. \tag{5}$$

Similarly to the classical Adaboost analysis we split the product of (5) to a term $\mathsf{w}_i^{(t)}$ already known from previous iterations and a term to be optimized in this iteration:

$$\bar{E} = \sum_{i=1}^{m} \mathsf{w}_i^{(t)} \cdot \left[(1 - \eta) \exp\{-y_i \alpha_t h_t(x_i)\} + \eta\right]. \tag{6}$$

This now suggests the following algorithm, called *Stragboost*, to successively minimize the expected exponential loss function in the presence of straggling classifiers.

Algorithm 1 *(Stragboost) For $t = 1, \ldots, M$, train $h_t(\cdot)$ to minimize*

$$\epsilon_t \triangleq \sum_{i=1}^{m} \mathsf{w}_i^{(t)} I(h_t(x_i) \neq y_i), \tag{7}$$

where

$$\mathsf{w}_i^{(t>1)} \triangleq \prod_{j=1}^{t-1} \left[(1 - \eta) \exp\{-y_i \alpha_j h_j(x_i)\} + \eta\right], \ \mathsf{w}_i^{(1)} \triangleq 1. \tag{8}$$

Then set

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right). \tag{9}$$

Proposition 2 *For i.i.d. straggling classifiers, at each iteration t of Algorithm 1, the data weights $\mathsf{w}_i^{(t)}$ and classifier coefficient α_t of h_t minimize the expected exponential loss*

$$\bar{E} \triangleq \sum_{A \subseteq [1,t]} P_\eta(A) \sum_{i=1}^{m} \exp\left\{-y_i \sum_{\substack{j=1 \\ j \in A}}^{t} \alpha_j h_j(x_i)\right\},$$

given the classifiers $h_1, \ldots, h_{t-1}$ and their coefficients $\alpha_1, \ldots, \alpha_{t-1}$ found in previous iterations.

Proof: (sketch) Extracting from (6) only the term that depends on h_t,α_t, we need to minimize $(1-\eta)\sum_{i=1}^{m} \mathsf{w}_i^{(t)} \cdot \exp\{-y_i\alpha_t h_t(x_i)\}$ which, by similar arguments to the ordinary Adaboost, reduces to training h_t with the weights $\mathsf{w}_i^{(t)}$ of (8) and setting α_t by (9). ∎

Proposition 2 implies that the presence of stragglers in the system changes the weighting of the data points during training (8) (effectively "slowing down" the training sequence), but leaves unchanged the formula that computes the classifier coefficients (9). Note that although there is no change in the formula, the final coefficients $\{\alpha_t\}_{t=1}^{M}$ will differ from ordinary Adaboost's because they depend on different classifiers and different data weights.

B. Re-weighting the non-straggling classifiers

Although the straggler identities are not known at training time, the fact that they are known at classification time allows re-weighting of their coefficients according to the instantaneous set A. We do this in the following.

Algorithm 2 *(Re-weighted Stragboost) Let* $\{h_t\}_{t=1}^{M}$ *be the classifiers trained in Algorithm* 1. *Given a subset* $A \subseteq [1,M]$ *of non-straggling classifiers, for* $t \in A$ *given in increasing order* $t_1, t_2, \ldots$, *set*

$$\mathsf{v}_i^{(t>t_1)} \triangleq \prod_{\substack{j<t \\ j\in A}} \exp\{-y_i\alpha'_j h_j(x_i)\},\ \mathsf{v}_i^{(t_1)} \triangleq 1, \qquad (10)$$

where

$$\epsilon'_t \triangleq \sum_{i=1}^{m} \mathsf{v}_i^{(t)} I(h_t(x_i) \neq y_i),\ \alpha'_t \triangleq \frac{1}{2}\ln\left(\frac{1-\epsilon'_t}{\epsilon'_t}\right). \qquad (11)$$

For Algorithm 2, the classifiers h_t are trained using the data weights $\mathsf{w}_i^{(t)}$ in Algorithm 1, but different data weights $\mathsf{v}_i^{(t)}$ are used to calculate a new coefficient α'_t for the classifier depending on the instantaneous set A. In practice, we run Algorithm 2 at training time with different subsets $A \subseteq [1,M]$, and store the resulting sets of coefficients $\{\alpha'_t\}_{t\in A}$ at the aggregator for use at classification time. It can be shown that α'_t in (11) minimizes the exponential loss function of the combined classifier $\sum_{\substack{j\leq t \\ j\in A}} \alpha'_j h_j(x)$, conditioned on the coefficients of the previous indices in A.

C. Empirical evaluation: straggling classifiers

We evaluate the performance of Stragboost with i.i.d. straggling classifiers. For the evaluation we use a synthetic dataset, generated from a depth-2 decision-tree model, and compare to the performance of ordinary Adaboost on the same dataset. In all the empirical results of this paper we use base classifiers based on decision stumps (depth-1 decision trees). We plot in Figure 1 the *average accuracy* results of the algorithms ($M = 8$) on the test set, as a function of the straggling probability η. Accuracy is the fraction of test-set data points that were correctly classified, and averaging is done over 25000 drawings of the set $A \subseteq [1,M]$. We can see that Stragboost (circle markers) offers a performance improvement over ordinary Adaboost (square markers), and the re-weighted version (Algorithm 2) improves even further. We also compare the performance to bagging, which is less affected by stragglers but in general inferior to boosting.

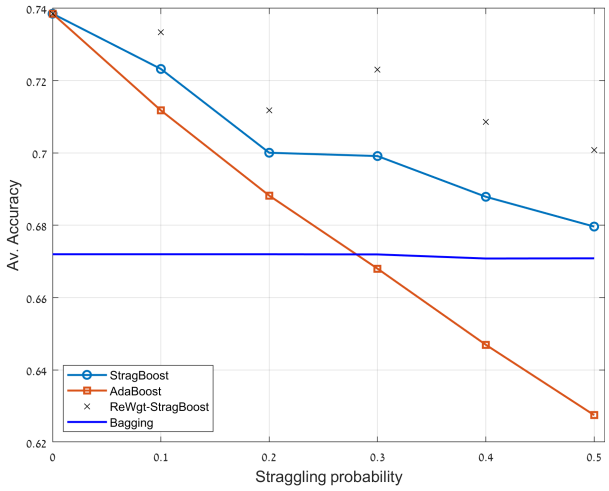

Fig. 1. Average accuracy as a function of straggling probability η for adaptive boosting classifiers, evaluated on synthetic data.

IV. Boosting with Flipping Classifiers

A straggling classifier considered in Section III delivers a "null" classification $\tilde{h}_t(x) = 0$ instead of the desired $\tilde{h}_t(x) = h_t(x)$. In this section we deal with classifiers that fail differently: they deliver a *flipped* classification $\tilde{h}_t(x) = -h_t(x)$. Intuitively, flipping classifiers are a more serious hindrance to accurate classification than straggling classifiers, because they silently contaminate the combined classifier with false inputs. In the *i.i.d. flipping* model, we assume that at classification time a base classifier flips with probability $\varepsilon < 0.5$, independently from other classifiers. We use the term "flip" – and not the more common term "error" – to distinguish these from the fact that $h_t(x)$ itself makes intrinsic classification errors. We extend the definition of the set A from Section III to contain the indices of the non-flipping classifiers in S. We denote by $P_\varepsilon(A)$ the probability that $(A, S\setminus A)$ is the partition of S to non-flipping,flipping classifiers. For the i.i.d. flipping model we have

$$P_\varepsilon(A) = (1-\varepsilon)^{|A|}\cdot\varepsilon^{|S|-|A|}. \qquad (12)$$

A simple variation of Algorithm 1 to the case of flipping classifiers gives the following algorithm, called *Flipboost*, that similarly minimizes the expected exponential loss function (formal result omitted).

Algorithm 3 *(Flipboost) identical to Algorithm* 1, *only replacing* $\mathsf{w}_i^{(t>1)}$ *in* (8) *by*

$$\mathsf{w}_i^{(t>1)} \triangleq \prod_{j=1}^{t-1} \left[\psi_j(x_i, y_i)\right], \qquad (13)$$

where

$$\psi_j(x,y) \triangleq (1-\varepsilon)\exp\{-y\alpha_j h_j(x)\} + \varepsilon\exp\{y\alpha_j h_j(x)\},$$

and (7) *by*

$$\epsilon_t = \sum_{i=1}^{m} \mathsf{w}_i^{(t)} \left[(1-\varepsilon)I(h_t(x_i)\neq y_i) + \varepsilon I(h_t(x_i)=y_i)\right]. \tag{14}$$

Note that unlike in the straggling case, the classifier coefficient α_t depends (through ϵ_t) on the flipping probability ε. If α_t is interpreted as the "confidence" of the combined classifier in $\tilde{h}_t$, then (14) implies that this confidence is reduced as ε grows.

A. Empirical evaluation: flipping classifiers

We use the same synthetic dataset from Section III-C and plot in Figure 2 the accuracy of Algorithm 3 (circle markers) in comparison to ordinary Adaboost (square markers), for $M = 8$. We also show the inferior performance of bagging on the same dataset.

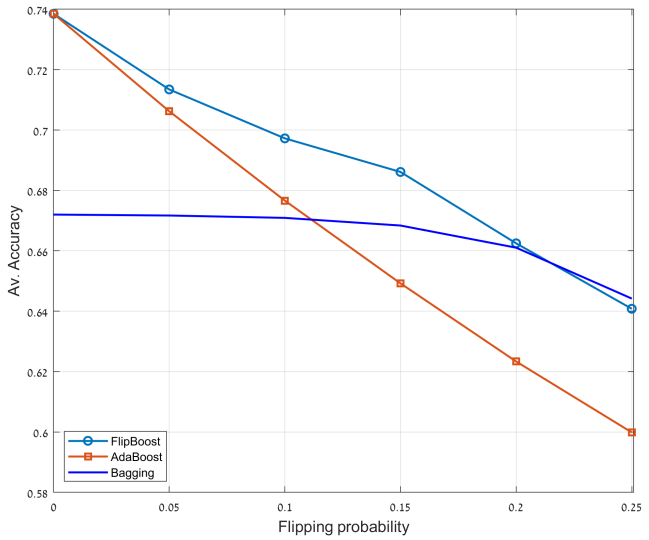

Fig. 2. Average accuracy as a function of flipping probability ε for adaptive boosting classifiers, evaluated on synthetic data.

V. Single-Error-Correcting Adaboost

So far in the paper, the mitigation of classifier errors (stragglers and flips) is done through minimizing the *expected* loss given the parameter of the i.i.d error source. While this is a natural optimization criterion, also showing some accuracy improvements empirically, it does not directly seek to maximize the accuracy of the aggregate classifier. Hence, in this section we pursue a more direct optimization of the error-prone classifier accuracy – for the special case of a *single erring classifier*. The same method can be extended to more than one error, with some increase in the training complexity (we discuss this extension in more detail later in the paper). Another difference from the preceding sections, is that we aim at high *worst-case* performance, that is, maximizing the minimum accuracy obtained under any single erring classifier. In the sequel we limit the discussion to flipping errors, but the results can be easily extended to straggling errors as well.

A. The 1-flip loss

Let $S(x) \triangleq \sum_{j=1}^{M} \alpha_j h_j(x)$, and define $S^{(t)}(x) \triangleq S(x) - 2\alpha_t h_t(x)$, which is the ensemble weighted sum when classifier t flips. We now define the *1-flip loss* of this ensemble.

Definition 1 *The **1-flip loss** of an ensemble* $\{h_j\}_{j=1}^{M}, \{\alpha_j\}_{j=1}^{M}$ *over a labeled dataset* $\mathcal{D} = \{(x_1,y_1),\ldots,(x_m,y_m)\}$ *is defined as*

$$\max_{t\in 1,\ldots,M} \left\{ \sum_{i=1}^{m} \exp\left(-y_i S^{(t)}(x_i)\right) \right\}. \tag{15}$$

When 1 classifier is allowed to flip arbitrarily frequently, the 1-flip loss replaces the usual exponential loss as a measure of ensemble quality. The t-th classifier has a dual role in suppressing the 1-flip loss: 1) trying to make $y_i S^{(t')}(x_i)$ as positive as possible for $t' \neq t$, and 2) not hurting $y_i S^{(t)}(x_i)$ too much when h_t flips. It is possible to extend Definition 1 to more than one flip, by taking the max over pairs, triples, quadruples, etc.

B. Reliability regularization of Adaboost

Recall (Proposition 1) that in Adaboost, each classifier's coefficient α_t minimizes the exponential loss given the previous classifiers and coefficients. When the t-th classifier flips, α_t also determines the magnitude by which the noisy weighted sum $S^{(t)}(x)$ differs from the reliable value $S(x)$. That suggests a *regularized* version of Adaboost for balancing the two effects of α_t.

Definition 2 *Given a positive real number* $\bar{\alpha}$*, the **reliability-regularized Adaboost** is identical to Adaboost, except in setting its coefficients to (cf.* (9)*)*

$$\bar{\alpha}_t = \min\left\{\bar{\alpha}, \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)\right\}. \tag{16}$$

By upper-bounding the classifiers' coefficients, the reliability-regularized Adaboost limits the effect of their flips, while potentially compromising the exploitation of their quality when they don't flip. Setting $\bar{\alpha}$ too high would lead a flipping classifier to cause frequent classification errors, while setting it too low would give poor minimization of the training error. Thus a very important question is: how to set $\bar{\alpha}$ to optimally balance these two considerations?

A useful tool toward picking the value of $\bar{\alpha}$ is the 1-flip loss of Definition 1. Given two possible values $\bar{\alpha}$,$\bar{\alpha}'$, we can run the regularized algorithm with each, and pick the one whose resulting ensemble achieves the lower 1-flip loss on the training set. Furthermore, evaluating the 1-flip loss on a sweep of $\bar{\alpha}$ values confirms its role as a balance between the quality and reliability of the ensemble; training on the CBCL dataset, the 1-flip loss exhibits *a U-shaped curve* as a function of $\bar{\alpha}$: first decreasing thanks to improving the quality of the

ensemble, and then starting to increase again due to stronger effects of single flips. This motivates an efficient method for finding the $\bar{\alpha}$ value minimizing the 1-flip loss.

C. A bisection algorithm for finding $\bar{\alpha}$

Minimizing the 1-flip loss over a sweep of $\bar{\alpha}$ values is a valid approach, but highly inefficient. For each value in the sweep we need to re-invoke the Adaboost algorithm, and many invocations are needed for a sufficiently fine search. In this subsection we propose a more efficient method based on a bisection algorithm. To allow bisecting the search space, we need a tool to answer the question of whether the current $\bar{\alpha}$ is *too high or too low*. For such a tool, we recall the relation (9) between α_t and the classifier error ϵ_t, and revisit a well-known identity [10] tying the exponential loss and the values of ϵ_t

$$\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i S(x_i)\right) = \prod_{j=1}^{M}\sqrt{1-4\left(\frac{1}{2}-\epsilon_j\right)^2}. \quad (17)$$

The relevant interpretation of (17) is that a set of $\{\alpha_j\}_{j=1}^{M}$ values imply (through $\{\epsilon_j\}_{j=1}^{M}$) a certain value of the full-ensemble's exponential loss. (The right-hand side of (17) may be more familiar as an upper bound on the training error.) That interpretation suggests the following idea. Suppose we replace the left-hand side of (17) with the 1-flip loss from (15), and at the right-hand side substitute the values $\bar{\epsilon}_t$ inverse mapped from the $\bar{\alpha}_t$ values from (16) (note that $\bar{\epsilon}_t$ do not correspond to real classifier errors). Then we change the equality to a hypothesis test, getting

$$\max_{t\in 1,\ldots,M}\left\{\frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i S^{(t)}(x_i)\right)\right\} \gtrless^{?} \prod_{j=1}^{M}\sqrt{1-4\left(\frac{1}{2}-\bar{\epsilon}_j\right)^2} \quad (18)$$

As we increase $\bar{\alpha}$, the right-hand side in general decreases thanks to larger $\bar{\alpha}_t$ values (smaller $\bar{\epsilon}_t$ values). The direction of the inequality in (18) can tell us about the current value of $\bar{\alpha}$. If the left-hand side is greater, that means the $\bar{\alpha}_t$ values are too high. Said differently, the high confidence we put in the classifiers does not pay in lowering the 1-flip loss accordingly. Conversely, a greater right-hand side suggests $\bar{\alpha}_t$ that are too low to predict the 1-flip loss, and thus can be increased.

We use (18) in a bisection algorithm that starts from unrestricted Adaboost ($\bar{\alpha}=\infty$), sets the initial search boundaries $\bar{\alpha}_{\max}=\max_{j=1}^{M}(\alpha_j)$, $\bar{\alpha}_{\min}=0$, and at each iteration halves the search interval downward or upward according to if the result of (18) is $>$ or $<$, respectively. The running time (and the number of training instances) of the bisection algorithm is significantly lower than the sweep, and it empirically reached final $\bar{\alpha}$ values that are within a few percentage points from the optimal value in the sweep.

D. Empirical evaluation: 1-flip loss minimization

On the CBCL face-detection dataset [8], we ran the bisection algorithm on the training set, and recorded the final $\bar{\alpha}$ for each $M\in\{8,\ldots,16\}$. We took the ensembles selected in the previous step and evaluated their accuracy on the test set, each time with a different index t flipping on all the data points. We evaluated the *bottom-3 accuracy*, which is the average of the 3 worst accuracies out of the M options of a single flipping classifier. We plot the resulting values in Figure 3 (circle markers), in comparison to ordinary Adaboost (square markers). While not every value of M sees improvement, the accuracy advantages are significant compared to the much smaller differences when Adaboost is a little better.

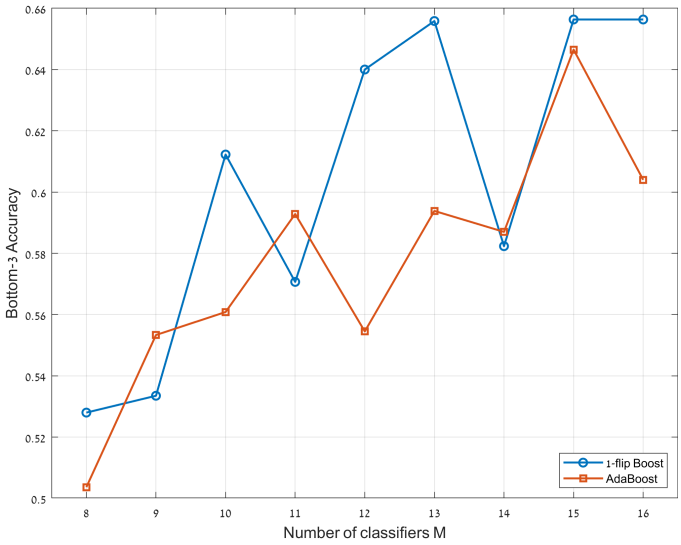

Fig. 3. Bottom-3 accuracy of proposed 1-flip algorithm vs. Adaboost.

VI. Discussion and Conclusion

Using boosted ensembles in distributed classification is highly promising, but introduces reliability issues. In the non-asymptotic case where M is fairly small, even a single flipping classifier can degrade the accuracy considerably if not mitigated. We proposed two approaches to address the issue within the boosting framework. The first (Stragboost, Flipboost) provides provable minimum-loss guarantees, but requires probabilistic assumptions on the errors. An interesting direction is to generalize the proposed algorithms to non-i.i.d. straggling/flipping models. The second approach (1-flip loss minimization+bisection) seeks high worst-case performance for a specified number of flipping classifiers. A potential improvement of the second method can be by using the 1-flip loss in an adaptive way within the training sequence itself, thus reducing the training complexity further. The challenge in doing so is that the 1-flip loss is inherently a global measure (1 out of M), and using it with different intermediate values $M'<M$ did not lead to good performance. More broadly, the problem of ensemble unreliability can be addressed by many other tools, for example by introducing some redundancy among the classifier outputs, and potentially detecting a flipping classifier prior to its aggregation.

REFERENCES

[1] T. Diettereich, "Ensemble methods in machine learning," *Multiple Classifier Systems, Springer, Berlin, Heidelberg*, pp. 1–15, Dec. 2000.

[2] Z. Xiao, Z. Luo, B. Zhong, and X. Dang, "Robust and efficient boosting method using the conditional risk," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3069–3083, Jul. 2018.

[3] Z. Wang, R. Schapire, and N. Verma, "Error adaptive classifier boosting (EACB): Leveraging data-driven training towards hardware resilience for signal inference," *IEEE Trans. Circuits Syst. I*, vol. 62, no. 4, pp. 1136–1145, Apr. 2015.

[4] Z. Wang, K. Lee, and N. Verma, "Overcoming computational errors in sensing platforms through embedded machine-learning kernels," *IEEE Trans. VLSI Syst.*, vol. 23, no. 8, pp. 1459–1470, Aug. 2015.

[5] Y. Kim, Y. Cassuto, and L. Varshney, "Distributed boosting classifiers over noisy channels," in *Proc. Asilomar Conference on Signals, Systems, and Computers*. IEEE, Nov. 2020.

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

[7] L. Breiman, "Bagging predictors," *Machine Learning, Springer, Berlin, Heidelberg*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[8] "CBCL FACE DATABASE," *Center for Biological and Computational Learning at MIT. Available: http://www.ai.mit.edu/courses/6.899/lectures/faces.tar.gz*, 2000.

[9] M. Frean and T. Downs, "A simple cost function for boosting," Department of Computer Science and Electrical Engineering, University of Queensland, Tech. Rep., 1998.

[10] R. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. Cambridge MA, USA: MIT Press, 2012.