Boosting the Federation: Cross-Silo Federated Learning without Gradient Descent

To appear in the proceedings of the International Joint Conference on Neural Networks (IJCNN 2022) , IEEE. Padova, Italy.

Boosting the Federation: Cross-Silo Federated Learning without Gradient Descent

1st Mirko Polato
Dept. of Computer Science
University of Turin
Turin, Italy
mirko.polato@unito.it

2nd Roberto Esposito
Dept. of Computer Science
University of Turin
Turin, Italy
roberto.esposito@unito.it

3rd Marco Aldinucci
Dept. of Computer Science
University of Turin
Turin, Italy
marco.aldinucci@unito.it

***Abstract*—Federated Learning has been proposed to develop better AI systems without compromising the privacy of final users and the legitimate interests of private companies. Initially deployed by Google to predict text input on mobile devices, FL has been deployed in many other industries. Since its introduction, Federated Learning mainly exploited the inner working of neural networks and other gradient descent-based algorithms by either exchanging the weights of the model or the gradients computed during learning. While this approach has been very successful, it rules out applying FL in contexts where other models are preferred, e.g., easier to interpret or known to work better.**

This paper proposes FL algorithms that build federated models without relying on gradient descent-based methods. Specifically, we leverage distributed versions of the AdaBoost algorithm to acquire strong federated models. In contrast with previous approaches, our proposal does not put any constraint on the client-side learning models. We perform a large set of experiments on ten UCI datasets, comparing the algorithms in six non-iidness settings.

***Index Terms*—federated learning, cross-silo, boosting, adaboost, ensemble learning**

I. Introduction

Recent years have been characterized by crucial advances in artificial intelligence and machine learning systems, by the widespread availability of massive computational resources, and by the availability of huge datasets. This unique conjunction of events allowed solving long-standing problems; for the first time in history, humanity witnessed the development of systems able to solve *intellectual* problems with human-level performances and beyond [1]. The consequent deployment of AI and ML methods throughout many industries has been a welcome innovation that generated, nonetheless, newfound concerns about the fairness of the results and the privacy of the involved data. Indeed, recent legislation in, e.g., Europe, United States and China have been enacted to strengthen the protection of user data used by AI and ML systems. On the other hand, companies tend to consider collected data as competing advantages and therefore are unwilling to share the data outside (sometimes even within different parts of) the organization. As a result, it is often the case that data is dispersed into many isolated islands, and ML practitioners are forbidden by laws and by the legitimate owners from collecting, fusing, and ultimately using the data to improve their systems. While protecting the privacy of users and the competing advantages of companies is arguably a fair objective, it nonetheless hampers the development of learning models that, by leveraging all the available data, could make a difference in the quality of life of many people who are subjected to the decisions made using AI systems.

Federated Learning (FL) has been proposed by McMahan et al. [2] as a way out of this conundrum, i.e., as a way to develop better AI systems without compromising the privacy of final users and the legitimate interests of private companies. Initially deployed by Google for predicting the text input on mobile devices, FL has been now adopted by many other industries such as mechanical engineering and health-care [3].

FL is a learning paradigm where multiple parties (clients) collaborate in solving a machine learning task using their private data under the coordination of an aggregator (a.k.a. server or coordinator). Each client's local data is not exchanged or transferred to any participant. The learning happens in rounds where model updates are computed by clients in insulation using local and private data, then aggregated on the server, then broadcast to the clients for the next round.

There are two main federated settings: cross-device and cross-silo. In cross-device FL, the parties can be edge devices (e.g., smart devices and laptops); they can be numerous (order of thousands or even millions). Parties are considered not reliable and with limited computational power. In the Cross-silo FL setting, the involved parties are instead organizations; the number of parties is limited, usually in the range [2, 100]. Given the nature of the parties, it can also be assumed that communication and computation are no real bottlenecks.

Since its introduction by McMahan et al., [2] Federated Learning mainly exploited the inner working of neural networks and other gradient descent-based algorithms by either exchanging the weights of the model or the gradients computed during learning. While this approach has been very successful, it rules out applying FL in contexts where other models would be preferred, either because they are more interpretable or known to work better.

This paper proposes a series of cross-silo FL algorithms for classification. These methods, which are based on ideas from AdaBoost [4]–[6] and distributed boosting literature [7], [8]

allow gradient-free federated learning. The algorithms pose minimal constraints on the learning settings of the clients, thus allowing a federation of models not specifically designed for FL, such as decision trees and SVMs. While there is no technical barrier to using our approach in cross-device federated learning settings, we have not conducted experiments to clarify the issue. Our intuition is that the approach will best work with reliable clients have many examples, and when communication cost is not high. We, therefore, believe that they are best suited for cross-silos settings and leave to future work investigating alternatives more kin to cross-device environments.

The main contributions of this paper are: *i)* we propose two new FL algorithms inspired by distributed AdaBoost literature, namely `DistBoost.F` and `PreWeak.F`; *ii)* we introduce a third algorithm (`AdaBoost.F`) purposely developed for FL; *iii)* we present a comprehensive evaluation of our solutions on ten UCI datasets and 6 data distribution settings.

For reproducibility purposes, all the code used to perform the experiments in this paper is available at *omitted for anonymity*.

II. Related Works

Ensemble Learning copes with the problem of strengthening the performances of a learning algorithm by iterating it and combining the results. Ensemble Learning is often employed by practitioners because it requires almost no parameters and can be used along with off-the-shelf algorithms to obtain strong models that are usually very robust to overfitting [6]. It is not surprising then that, at the beginning of this century a large swat of research has been devoted to the topic and that many flavors of ensemble learning have been proposed during those years (e.g., Bagging [9], Boosting and its variants [5], Stacking [10], ECOC [11], etc.). In this context, the original boosting algorithm from Schapire [12] is fundamental because by constructively solving the weak learnability problem [12] spawned massive interest in the field and posed the basis for the development of AdaBoost [5], arguably the best-known algorithm in the field. The main idea in Schapire's boosting algorithm [12], and hence in AdaBoost, is that by leveraging a distribution over the examples one is able force the weak learning algorithm to focus on specific portions of the examples space. This can be then used to drive down the error of the ensemble exponentially fast. AdaBoost appears particularly interesting as a candidate tool for FL, as it effectively combines classifiers which may be learned independently by the FL clients. Furthermore, it could be argued that, as long as at least one of the clients can find a model which is slightly better than the random guess over the complete dataset, AdaBoost should be able to drive the error of the ensemble on the training set to its theoretical minimum no matter other factors (such as the possible non-iidness of the data distribution).

Most of the FL literature focuses on gradient-based methods with very few exceptions. [13] proposes Federated Forest, a lossless federated version of the classical Random Forest (RF) algorithm for vertically partitioned data. In this method, trees are built on node splits selected by the aggregator that repeatedly asks clients for the impurity index and picks the minimum. Federated Forest guarantees privacy preservation mainly using features/labels' encoding. However, label encoding may fail in the case of binary classification tasks. A very different approach to learning RFs is presented in [14] where the federation is managed using Blockchain technology that guarantees security even against adversarial participants. Vertical FL (VFL) is the learning setting in [15] that presents federated algorithm for classification/regression trees based on Multi-Party Computation [16]. The authors also describe possible extensions of the methodology to gradient-boosting trees and linear regression. In [17], the VFL setting is considered in the context of kernel-based methods. The authors propose a privacy-preserving protocol to build dot-product kernel matrices, showing the technique's effectiveness on top-N recommendation tasks. To the best of our knowledge, our paper is the first to propose a federated version(s) of AdaBoost where the (weak) classifiers can be induced by any learning algorithm.

As briefly mentioned in the introduction, two of the algorithms presented in this paper are based on a distributed version of AdaBoost, namely DistBoost [7] and PreWeak [8] that we will describe in Section III. In [18], a distributed agnostic boosting algorithm is described. Differently from AdaBoost, the method uses a non-exponential multiplicative weight update rule that is further adjusted using the *Bregman projection*. Here, we propose a federated adaptation of AdaBoost, and we would argue that a similar methodology may also apply to the approach in [18].

Boosting-based FL has been little studied in the literature. All published works on the topic focus on gradient-boosting trees [19], [20] and most of them are designed for vertically partitioned data [21]–[24]. Homomorphic encryption and secret sharing schemes are used to guarantee privacy, with the only exception of [19], [21] that use a differential private approach. The cross-silo setting is considered in both [21] and [24] (decentralized FL).

We differentiate from these previous works because our federated boosting algorithms can be used with any weak learner, and our setting is horizontal FL.

III. Ensemble Learning based Federated Learning

A. Notation

This section summarizes the notation used throughout Section III. Vectors are indicated with lowercased bold letters, e.g., $\mathbf{x}$ and, when not specified differently, they are meant to be real column vectors. An element of a vector $\mathbf{x}$ (that is, a number) is indicated with x_i. We also use the notation $[x_i]_{i=1}^n$ to indicate a n-dimensional vector in which the elements are x_i with i in the range $[1, n]$. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x} \parallel \mathbf{y}$ indicates their concatenation and, accordingly, the notation $\|_{i=1}^n \mathbf{x}_i$ denotes the concatenation of vectors $\mathbf{x}_1 \dots \mathbf{x}_n$. We refer to matrices with bold uppercased letters, like $\mathbf{X}$. We use the notation $[\![x]\!]$ to represent the indicator function, i.e., $[\![P]\!] = 1$ if the predicate

P is true, 0 otherwise. We denote with C the number of clients, with $\mathbf{X}_c$ the portion of data owned by the c-th client and with $\mathbb{X} = \bigcup_{c\in\{1\dots C\}} \mathbf{X}_c$ the dataset that would be obtained by joining all parties' datasets.

In the algorithms, we will use the function $\textbf{send}_x(\text{aggr}, x)$ (client-side) to indicate a message that carries x sent from a client to the aggregator. On the aggregator side, we use the function $\textbf{broadcast}_x(x)$ that sends x to all clients. To each $\textbf{send}_x(\text{aggr}, x)$ or $\textbf{broadcast}_x(x)$ from a sender corresponds a $\textbf{receive}_x(s)$ in the receiver, where s identifies the sender. In all expressions involving receiving from all clients, we assume that the underlying communication run-time asynchronously and non-deterministically receives one message from any client until all messages has been received.

B. Algorithms

This section introduces `DistBoost.F` (Sec.III-B1) and `PreWeak.F` (Sec.III-B2), i.e., the adaptation to the FL setting of the distributed boosting algorithms DistBoost [7] and PreWeak [8], as well as a novel algorithm: `AdaBoost.F` (Sec.III-B3). Here and in the following, we shall refer to the DistBoost and PreWeak as the inspiring algorithms for our proposals. Although it should be mentioned the proposed versions also deal with multi-class instead of binary problems.

Since we work in a cross-silo FL setting, we assume that the clients are reliable and have enough computational power as well as a stable and secure connection [25], [26]. With these assumptions, our proposals expect a certain degree of synchronicity between the clients and the aggregator. However, all the proposed techniques can easily handle clients' failure, for instance, by using a timeout on the clients that exclude their participation from that federated round.

In [5] Freund and Schapire formally proved that, provided that the weak learner can induce a decision rule which is consistently better than random guessing, AdaBoost reduces the ensemble error over the training set exponentially fast in the number T of the combined weak models. It is worth emphasizing that this is the only constraint posed by the algorithm. As shown by Freund and Schapire [4], this holds true even when the weak learner behaves adversarially towards the ensemble learner. While this is not relevant in most scenarios, in the federated learning case, the weak learners only work with a subset of the available data. In a sense, it can be thought that malevolent learners try to make the ensemble learner fail on the part of the data (the data they do not own). This argument shows that, as long as at least one client can produce a model better than random guess over the entire dataset, a distributed version of AdaBoost, modified to guarantee that no information about the local dataset is exchanged, should be able to drive the ensemble error to its minimum exponentially fast. This is the main idea on the basis of our work.

1) `DistBoost.F`: DistBoost [7] was designed to run on distributed clients who can communicate with each other. The data is partitioned across the clients, and each of them stores a local weight distribution. The core idea of DistBoost is that the union of the local distributions represents a good approximation of the centralized AdaBoost's example distribution. The training phase of DistBoost is similar to the one of AdaBoost, but it happens in a distributed manner. At each iteration, a new weak hypothesis is learned from each client. Unlike AdaBoost, the "global" weak hypothesis is the committee of all the weak classifiers locally trained by the clients. Then, weights are updated according to the AdaBoost update rule, and a new iteration begins. To make this last step possible, each client must communicate to the other clients the errors the weak global hypothesis commits to its own local data. To simplify and mimic the centralized FL setting, we can assume a server acts as an intermediary. The final classifier is the weighted sum of the (global) weak classifiers, i.e., the committee of the weak local classifiers.

Before describing the federated adaptation of DistBoost, we have to modify the algorithm to make it able to solve multi-class problems. According to the SAMME algorithm [27], the only change AdaBoost (and hence DistBoost) needs to manage more than two classes is in the way the α terms are computed. Specifically, the computation should be updated as it follows:

$$\alpha^t = \log\left(\frac{1-\epsilon^t}{\epsilon^t}\right) + \log(K-1), \tag{1}$$

where t is the current iteration, α^t is the weight of the weak hypothesis h^t, K is the number of classes, and ϵ^t is the weighted error committed by h^t.

Algorithms 1 and 2 describe the pseudo-code of the aggregator and the client-side of `DistBoost.F`. To initiate the federated training, the clients send to the aggregator the size of the local training set that will be needed to compute the weighted error (ϵ^t) and hence the weight of the weak hypothesis (α^t). Then, for each federated round, clients train a weak classifier according to the current local examples' distribution $\mathbf{d}$ (initially uniform) and send their current weak classifier to the aggregator. The aggregator receives all the weak classifiers and creates the "global" weak classifier $\mathbf{h}^{t\star}$ (as in DistBoost) and broadcasts it to the clients. Afterward, each client computes the local error of $\mathbf{h}^{t\star}$ and sends it back to the aggregator. The aggregator thus computes α^t (see Eq. (1)) using the mean over all the received local errors. Finally, the aggregator broadcasts α^t to the clients that can update their local distribution, and a new federated round can start.

2) `PreWeak.F`: As shown in [8], PreWeak represents, in empirical terms, an improvement over DistBoost. However, while the downside of PreWeak is the communication cost, it could be easily shown that its federated adaptation (`PreWeak.F`) has a communication cost that is asymptotically not higher than `DistBoost.F`'s. As of `DistBoost.F`, `PreWeak.F` is a federated adaptation of PreWeak for multi-class problems. The adjustments to make `PreWeak.F` able to handle multi-class tasks are the same as for `DistBoost.F`. The main idea behind PreWeak [8] is to reduce the overfitting of DistBoost by pre-building a large set of weak learners that can be tested on the global distribution. So, the core difference between DistBoost and PreWeak lies in how the

Algorithm 1: `DistBoost.F` (aggregator)

Input: C: number of clients
T: dimension of the ensamble
K: number of classes
Output: $\text{ens}(\mathbf{x}) \triangleq \text{vote}([h^{t\star}]_{t=1}^{T}, [\alpha^t]_{t=1}^{T}, \mathbf{x})$

1 $N \leftarrow \sum_{c=1}^{C} \textbf{receive}_n(c)$
2 **for** $t \in \{1 \dots T\}$ **do**
3 $\quad \mathbf{h}^t \leftarrow [\textbf{receive}_h(c)]_{c=1}^{C}$
4 $\quad h^{t\star}(\mathbf{x}) \triangleq \sum_c h_c^t(\mathbf{x})$
5 $\quad \textbf{broadcast}_{\mathbf{h}}(\mathbf{h}^t)$
6 $\quad \epsilon^t \leftarrow \frac{1}{N} \sum_{c=1}^{C} \textbf{receive}_\epsilon(c)$
7 $\quad \alpha^t \leftarrow \log\left(\frac{1-\epsilon^t}{\epsilon^t}\right) + \log(K-1)$
8 $\quad \textbf{broadcast}_\alpha(\alpha^t)$
9 $\textbf{broadcast}_{\text{stop}}(\text{stop})$

Algorithm 2: `DistBoost.F` (client)

Input: $\mathcal{A}$: weak learner
$\mathbf{X} \in \mathbb{R}^{n \times m}$: training data
$\mathbf{y} \in \{1, \dots, K\}^n$: training labels

1 $\textbf{send}_n(\text{aggr}, n)$
2 $\mathbf{d} \leftarrow \mathbf{1}$
3 **while** *not stop* **do**
4 $\quad h \leftarrow \mathcal{A}\left(\mathbf{X}, \mathbf{y}, \frac{\mathbf{d}}{\|\mathbf{d}\|_1}\right)$
5 $\quad \textbf{send}_h(\text{aggr}, h)$
6 $\quad \mathbf{h} \leftarrow \textbf{receive}_{\mathbf{h}}(\text{aggr})$
7 $\quad \epsilon \leftarrow \sum_{c=1}^{|\mathbf{h}|} \mathbf{d}^\top [\![\mathbf{y} = h_c(\mathbf{X})]\!]$
8 $\quad \textbf{send}_\epsilon(\text{aggr}, \epsilon)$
9 $\quad \alpha^\star \leftarrow \textbf{receive}_\alpha(\text{aggr})$
10 $\quad \mathbf{d} \leftarrow \left[d_i \exp\left(-\alpha^\star(-1)^{[\![\text{vote}(\mathbf{h},\mathbf{1},\mathbf{x}_i) \neq y_i]\!]}\right)\right]_{i=1}^{n}$

weak classifiers are learned. Specifically, PreWeak separately learns on each client an AdaBoost classifier (i.e., the T weak models). Then, in turn, the aggregator trains a new ensemble using a slightly modified AdaBoost which uses as hypothesis space the union of the sets of the weak classifiers learned by the clients.

The adaptation of PreWeak to the federated setting is easier than DistBoost's. Up to the aggregator's training phase, `PreWeak.F` is the same as PreWeak (lines 1-2 in Alg. 3 and lines 1-3 in Alg. 4) with the addition of the broadcast of all the weak classifiers ($\mathbf{h}^+$) to the clients. However, the training on the aggregator is very different from PreWeak because the distribution over the examples is unknown to the aggregator. To deal with it, the aggregator must coordinate with the clients (*for* loop in Algorithm 3) to recompute at each round the error and the distribution. This coordination is carried out in two main steps: (i) clients compute the error of all the weak classifiers according to the current local distribution and send them back to the aggregator; (ii) the aggregator receives the errors, averages them and identifies the best weak classifier ($h_{j^\star}^+$). Then, the aggregator computes and broadcasts α^t along with the id of the best hypothesis ($j^\star$). Finally, the clients update their local distribution, and a new federated round can start.

Algorithm 3: `PreWeak.F` (aggregator)

Input: C: number of clients
T: dimension of the ensamble
K: number of classes
Output: $\text{ens}(\mathbf{x}) \triangleq \text{vote}([h_{j^{t\star}}]_{t=1}^{T}, [\alpha^t]_{t=1}^{T}, \mathbf{x})$

1 $N \leftarrow \sum_{c=1}^{C} \textbf{receive}_n(c)$
2 $\mathbf{h}^+ \leftarrow \|_{c=1}^{C} \textbf{receive}_{\mathbf{h}}(c)$ ▷ $C \times T$ `classifiers list`
3 $\textbf{broadcast}_{\mathbf{h}^+}(\mathbf{h}^+)$
4 **for** $t \in \{1 \dots T\}$ **do**
5 $\quad \boldsymbol{\epsilon}^t \leftarrow \sum_{c=1}^{C} \textbf{receive}_{\boldsymbol{\epsilon}}(c)$ ▷ $C \times T$ `errors list`
6 $\quad j^{t\star} \leftarrow \arg\min_j \epsilon_j^t$
7 $\quad \epsilon^{t\star} \leftarrow \frac{1}{N} \epsilon_{j^{t\star}}^t$
8 $\quad \alpha^t \leftarrow \log\left(\frac{1-\epsilon^{t\star}}{\epsilon^{t\star}}\right) + \log(K-1)$
9 $\quad \textbf{broadcast}_\alpha(\alpha^t)$
10 $\quad \textbf{broadcast}_j(j^{t\star})$
11 $\textbf{broadcast}_{\text{stop}}(\text{stop})$

Algorithm 4: `PreWeak.F` (client)

Input: $\mathcal{A}$: weak learner
$\mathbf{X} \in \mathbb{R}^{n \times m}$: training data
$\mathbf{y} \in \{1, \dots, K\}^n$: training labels
T: number of local models to be learned

1 $\textbf{send}_n(\text{aggr}, n)$
2 $\mathbf{h} \leftarrow \text{AdaBoost}(\mathbf{X}, \mathbf{y}, \mathcal{A}, T)$ ▷ T `AdaBoost's models`
3 $\textbf{send}_{\mathbf{h}}(\text{aggr}, \mathbf{h})$
4 $\mathbf{h}^+ \leftarrow \textbf{receive}_{\mathbf{h}^+}(\text{aggr})$
5 $\mathbf{d} \leftarrow \mathbf{1}$
6 **while** *not stop* **do**
7 $\quad \boldsymbol{\epsilon} \leftarrow \left[\sum_{i=1}^{n} d_i [\![h_j^+(\mathbf{x}_i) \neq y_i]\!]\right]_{j=1}^{|\mathbf{h}^+|}$
8 $\quad \textbf{send}_{\boldsymbol{\epsilon}}(\text{aggr}, \boldsymbol{\epsilon})$
9 $\quad \alpha^\star \leftarrow \textbf{receive}_\alpha(\text{aggr})$
10 $\quad j^\star \leftarrow \textbf{receive}_j(\text{aggr})$
11 $\quad \mathbf{d} \leftarrow \left[d_i \exp\left(-\alpha^\star(-1)^{[\![h_{j^\star}^+(\mathbf{x}_i) \neq y_i]\!]}\right)\right]_{i=1}^{n}$

3) `AdaBoost.F`: Algorithms 5 and 6 describe the `AdaBoost.F` algorithm which, in contrast to `DistBoost.F` and `PreWeak.F`, is not based on a previously known distributed AdaBoost algorithm.

The differences between these two algorithms and `AdaBoost.F` can be cast in terms of the hypothesis space they search. In the case of `DistBoost.F` models are the average of weak classifiers learned on the clients. The important point to notice is that such models will not be able to aggressively fit the nuances of a particular dataset. It will then be likely

that they work best when the examples are uniformly distributed between clients. `PreWeak.F`, on the contrary, select models from a pool of weak classifiers that AdaBoost have aggressively fitted on the specific nuances of the local dataset. The possible problem here is that those classifiers might be too specific and `PreWeak.F` cannot find hypotheses that do not deteriorate the overall performances. `AdaBoost.F` mediates between these two extremes by allowing the clients to learn a single model per round and keeping only the best one. In this way, all the local weak learners always have the information needed to optimize for the current distribution of the weights (which is not true for `PreWeak.F`) and there is no averaging of models hindering specialization on the local data. The hypothesis space searched by `AdaBoost.F` should then contain models that are not as specific as those used by `PreWeak.F` and they are also not averaged between all clients as in the case of `DistBoost.F`. Which strategy is better is likely to be situational. We investigate the behavior of the algorithms and their trade-off in Section IV and spend the rest of this section introducing `AdaBoost.F` in more details.

`AdaBoost.F` requires each client c to train a classifier based on its own $\mathbf{X}_c$ dataset and the current weight distribution that each client maintains locally. The C models induced in this way are collected into a vector $\mathbf{h}$ by the aggregator (line 3) and immediately broadcast to all the clients (line 4) so to provide them with the complete set of models learned during the round. Then, each client computes the vector

$$\boldsymbol{\epsilon} = \left[\mathbf{d}^\top [\![\mathbf{y} = h_c(\mathbf{X})]\!]\right]_{c=1}^{|\mathbf{h}|}$$

containing the weighted error committed by models in $\mathbf{h}$. $\boldsymbol{\epsilon}$ vectors are then collected by the aggregator into a matrix $\mathbf{E}$ and the index $c^{t\star}$ of the best model can be then computed as:

$$c^{t\star} \leftarrow \arg\min_c \sum_{c'=1}^{C} \mathbf{E}^t_{cc'},$$

where the summation over c' computes each model's weighted error over $\mathbb{X}$. Then α is calculated, and the information needed by the clients to update their local copy of the weight distribution is broadcast.

IV. Experiments

A. Experimental setting

We compared our federated algorithms, namely `DistBoost.F`, `PreWeak.F`, and `AdaBoost.F`, with the centralized algorithm SAMME [27] (multiclass AdaBoost). For all methods, we fixed the number of weak learners (federated rounds) $T = 300$. As weak learners, we employed Decision Trees with up to 10 leaves (as in [27]). However, the proposed algorithms are agnostic to the choice of the weak learner. The simulated federation contains 10 clients, which is a standard choice [26] in the cross-silo setting. We assumed that all clients correctly participated in all rounds during the simulation.

Algorithm 5: `AdaBoost.F` (aggregator)

Input: C: number of clients
T: dimension of the ensamble
K: number of classes
Output: $\text{ens}(\mathbf{x}) \triangleq \text{vote}([h^{t\star}]_{t=1}^T, [\alpha^t]_{t=1}^T, \mathbf{x})$

1 $N \leftarrow \sum_{c=1}^{C}$ **receive**$_n(c)$
2 **for** $t \in \{1 \dots T\}$ **do**
3 $\quad \mathbf{h}^t \leftarrow [\textbf{receive}_h(c)]_{c=1}^C$
4 $\quad$ **broadcast**$_{\mathbf{h}}(\mathbf{h}^t)$
5 $\quad \mathbf{E}^t \leftarrow [\textbf{receive}_{\boldsymbol{\epsilon}}(c)]_{c=1}^C$ ▷ $C \times C$ errors matrix
6 $\quad c^{t\star} \leftarrow \arg\min_c \sum_{c'=1}^{C} \mathbf{E}^t_{cc'}$
7 $\quad \epsilon^{t\star} \leftarrow \frac{1}{N} \sum_{c=1}^{C} \mathbf{E}^t_{cc^{t\star}}$
8 $\quad \alpha^t \leftarrow \log\left(\frac{1-\epsilon^{t\star}}{\epsilon^{t\star}}\right) + \log(K-1)$
9 $\quad$ **broadcast**$_\alpha(\alpha^t)$
10 $\quad$ **broadcast**$_c(c^{t\star})$
11 **broadcast**$_{\text{stop}}$(stop)

Algorithm 6: `AdaBoost.F` (client)

Input: $\mathcal{A}$: weak learner
$\mathbf{X} \in \mathbb{R}^{n \times m}$: training data
$\mathbf{y} \in \{1, \dots, K\}^n$: training labels

1 **send**$_n$(aggr, n)
2 $\mathbf{d} \leftarrow \mathbf{1}$
3 **while** *not stop* **do**
4 $\quad h \leftarrow \mathcal{A}\left(\mathbf{X}, \mathbf{y}, \frac{\mathbf{d}}{\|\mathbf{d}\|_1}\right)$
5 $\quad$ **send**$_h$(aggr, h)
6 $\quad \mathbf{h} \leftarrow$ **receive**$_{\mathbf{h}}$(aggr)
7 $\quad \boldsymbol{\epsilon} \leftarrow \left[\mathbf{d}^\top [\![\mathbf{y} = h_c(\mathbf{X})]\!]\right]_{c=1}^{|\mathbf{h}|}$
8 $\quad$ **send**$_{\boldsymbol{\epsilon}}$(aggr, $\boldsymbol{\epsilon}$)
9 $\quad \alpha^\star \leftarrow$ **receive**$_\alpha$(aggr)
10 $\quad c^\star \leftarrow$ **receive**$_c$(aggr)
11 $\quad \mathbf{d} \leftarrow \left[d_i \exp(-\alpha^\star(-1)^{[\![h_{c^\star}(x_i) \neq y_i]\!]})\right]_{i=1}^n$

TABLE I: Statistics of the datasets used in the experimentation

Dataset	Train	Test	# labels	# feat.
`adult`	30,162	15,060	2	14
`kr-vs-kp`	2,557	639	2	36
`forestcover`	250,000	245,141	2	54
`splice`	2,552	638	3	61
`vehicle`	677	169	4	18
`segmentation`	209	2,099	7	19
`sat`	4,435	2,000	8	36
`pendigits`	7,494	3,498	10	16
`vowel`	792	198	11	27
`letter`	16,000	4,000	26	16

We evaluated the methods on 10 benchmark datasets from the UCI [28] repository. The details of the datasets are summarized in Table I.

We tried to select diverse datasets, especially w.r.t. the number of labels and the number of instances. To replicate the same test as in [7], we converted `forestcover` into a binary classification task by keeping only the examples of classes 1 and 2. When available, we used UCI's training and test division; otherwise, we randomly split the dataset using an 80/20 training and test proportion. As already stated, for `forestcover` we used the same setting as in [7] (i.e., a 50/50 split). All datasets have been preprocessed using the same procedure: (i) instances with missing values are removed; (ii) categorical features are one-hot encoded. The datasets have been distributed across the clients using the procedures described in Section IV-B.

The methods have been compared using standard classification metrics like accuracy, precision, recall, and F1. For space reasons, we only report the F1 score, which is the harmonic mean of the precision and recall, and it considers how the data is distributed.

Each experiment has been repeated five times. The reported results are the averages (with their standard deviation) over these runs. The python implementation of the methods and their evaluation is available at *omitted for anonimity*.

B. Non-iidness

All federated methods have been tested on different data distributions across clients to test their behavior in case of non-iidness. Besides the iid case (uniform data distribution), we also consider the following types of non-iidness: quantity skew, prior shift (three versions), and covariate shift [26]. In the following, we briefly describe how we implemented them. It is worth noticing that we guarantee that each client has at least two examples of two different classes in every type of skewness.

Quantity skew. In quantity skew, the size of the local dataset varies across parties. To achieve this imbalance, given N parties, we allocate to each party a proportion of the examples according to the Power distribution. Specifically, we sample $p \sim \text{Pow}_N(\alpha)$ with shape parameter $\alpha > 0$, and we assign to client j a p_j proportion of the examples. In this way, the dimension of each local dataset across the users follows a power-law distribution. In the experiments, we fixed $\alpha = 4$.

Label quantity skew. In this version of the prior shift, each party owns data samples of a fixed number of label [29]. For all datasets but `letter` we set to 2 the number of labels per client. For `letter`, we fixed it to 3 because the dataset has more than 20 classes.

Dirichlet labels skew. This is another example of prior shift. The label imbalance is simulated allocating to each party a proportion of the examples according to the Dirichlet distribution as suggested in [29]. Specifically, we sample $p_k \sim \text{Dir}_N(\beta)$ and allocate a $p_{k,j}$ proportion of the instances of class k to party j. β is the concentration parameter ($\beta > 0$) of the Dirichlet distribution. Values of $\beta \to 0$ led to unbalanced partitioning. In the experiments we fixed $\beta = 0.5$ as in [29].

Pathological labels skew. This is the third example of the prior shift. According to [2], the following procedure creates a pathological labels imbalance. First, we sort the data by label (encoded as an integer in $[1, K]$). Then, we divide the dataset into m shards of the same dimension and assign to each client m' shards. Most clients will only have examples of m' classes with this partitioning. In the experiments we fixed $m' = 3$ and $m = N \times m'$, where n is the number of clients. Note that using $m' = 3$ prevents having a skewness that is too close to the *label quantity skew*.

Covariate shift. This type of non-iidness can also be called a feature distribution skew. Here, the feature distributions $P(\mathbf{x}_i)$ vary across parties although the $P(y_i|\mathbf{x}_i)$ is the same. To achieve this, we design the following procedure. In turn, we take the example of a single class and extract the first principal component (through PCA). We divide the examples according to this component in m percentiles that we call *modes*. Then, only examples from one mode are selected uniformly at random for each user. In this way, we try to ensure that $P(y_i|\mathbf{x}_i)$ is the same across the clients. In [29] the same skewness has been created, adding a different level of noise in the data of each client. This particular procedure was not suitable for our purposes.

C. Results

We start by investigating how beneficial are the federations built by the proposed algorithm. To do that, we need to evaluate the performance of a possible competitor built only on local data. Then, for each non-iidness type, we ran the SAMME algorithm on each client, using only the local data for training and recording the F1 score over a fixed independent test set. Figure 1 shows, for all the clients and all the data distributions, the difference in F1 score (ΔF1) between the local run of SAMME (local SAMME in the following) and the best achieved by one of the federated algorithms we described. The lower (more negative) ΔF1 is for a given point; the more beneficial is the federation for the corresponding client and data distribution setting. Results are presented only for the `pendigits` dataset due to space limitations.

The first thing to notice is that participating in the federation is generally beneficial to all clients, especially in a non-iid data distribution. The only exception is in the quantity skew scenario where the clients with many examples (the head of the power-law) can reach F1 scores even higher than the federation. This is reasonable because the client is close to having all the available data; it runs then in a setting similar to running SAMME over the fused dataset that is generally better than its federated counterpart. We can also observe that the scenarios with a prior shift (i.e., Labels Quantity, Dirichlet, and Pathological) are the most challenging ones. This is particularly apparent for the label quantity skew and the pathological label skew where, by design, we assign only a small subset of labels per client. Notice that *in absolute terms* the performances of local SAMME on the label quantity

	Dataset	Model	Uniform	Quantity	Cov. Shift	Dirichlet	Pathological	Labels Quantity
binary datasets	adult	Samme	86.45 ± 0.13	86.45 ± 0.13	86.45 ± 0.13			
		DistBoost.F	84.71 ± 0.15	84.66 ± 0.39	84.64 ± 0.24			
		PreWeak.F	84.97 ± 0.16	85.24 ± 0.07	**85.28 ± 0.04**			
		AdaBoost.F	**85.58 ± 0.06**	**86.01 ± 0.14**	85.12 ± 0.10			
	forestcover	Samme	81.64 ± 0.34	81.26 ± 0.40	81.70 ± 0.28			
		DistBoost.F	83.27 ± 0.20	82.10 ± 0.20	76.90 ± 1.48			
		PreWeak.F	**84.46 ± 0.18**	**84.09 ± 0.26**	**83.70 ± 0.13**			
		AdaBoost.F	83.67 ± 0.21	83.47 ± 0.14	79.42 ± 0.60			
	kr-vs-kp	Samme	99.75 ± 0.21	99.75 ± 0.21	99.75 ± 0.21			
		DistBoost.F	98.84 ± 0.36	99.09 ± 0.49	92.94 ± 2.58			
		PreWeak.F	97.78 ± 0.86	96.50 ± 0.79	95.16 ± 1.31			
		AdaBoost.F	**99.38 ± 0.29**	**99.41 ± 0.17**	**95.94 ± 0.56**			
multi-class datasets	splice	Samme	95.74 ± 0.63	95.74 ± 0.63	95.74 ± 0.63	95.74 ± 0.63	95.74 ± 0.63	95.74 ± 0.63
		DistBoost.F	94.67 ± 0.79	93.48 ± 1.43	94.26 ± 0.45	92.92 ± 0.99	91.41 ± 5.93	93.64 ± 0.76
		PreWeak.F	94.89 ± 1.03	94.64 ± 0.72	94.67 ± 0.73	**94.86 ± 0.67**	94.15 ± 1.02	**96.08 ± 1.01**
		AdaBoost.F	**95.61 ± 0.62**	**95.67 ± 0.81**	**94.83 ± 1.01**	94.51 ± 0.95	**94.17 ± 0.98**	94.70 ± 0.55
	vehicle	Samme	74.47 ± 1.48	74.47 ± 1.48	74.47 ± 1.48	74.47 ± 1.48	74.47 ± 1.48	74.47 ± 1.48
		DistBoost.F	68.82 ± 2.53	68.94 ± 4.17	66.35 ± 5.12	66.82 ± 2.22	67.65 ± 3.72	55.76 ± 8.03
		PreWeak.F	72.24 ± 3.66	**72.24 ± 3.98**	70.00 ± 4.83	**71.88 ± 5.37**	**70.47 ± 4.29**	**68.12 ± 3.26**
		AdaBoost.F	**72.94 ± 3.40**	69.88 ± 3.82	**70.82 ± 4.02**	69.76 ± 3.54	68.47 ± 1.29	65.29 ± 5.38
	segmentation	Samme	95.09 ± 0.20	95.09 ± 0.20	95.09 ± 0.20	95.09 ± 0.20	95.09 ± 0.20	95.09 ± 0.20
		DistBoost.F	85.91 ± 1.62	85.58 ± 2.31	82.46 ± 3.20	81.20 ± 3.84	**81.80 ± 2.25**	**54.84 ± 7.60**
		PreWeak.F	**87.55 ± 1.28**	**87.60 ± 1.74**	**87.42 ± 1.55**	**86.70 ± 3.69**	81.34 ± 5.54	38.70 ± 13.44
		AdaBoost.F	86.07 ± 2.86	87.34 ± 2.87	85.38 ± 1.48	83.01 ± 3.69	70.36 ± 8.81	49.95 ± 10.88
	sat	Samme	85.14 ± 0.34	85.14 ± 0.34	85.14 ± 0.34	85.14 ± 0.34	85.14 ± 0.34	85.14 ± 0.34
		DistBoost.F	81.78 ± 1.61	81.18 ± 0.71	82.08 ± 1.76	81.39 ± 1.53	78.54 ± 5.79	48.82 ± 6.14
		PreWeak.F	**86.41 ± 0.69**	**85.26 ± 1.57**	**85.87 ± 0.31**	**85.20 ± 0.15**	**82.65 ± 4.97**	**66.61 ± 8.97**
		AdaBoost.F	83.52 ± 0.58	83.79 ± 1.32	82.58 ± 0.50	81.56 ± 0.90	77.01 ± 5.17	55.18 ± 7.62
	pendigits	Samme	94.56 ± 0.41	94.56 ± 0.41	94.56 ± 0.41	94.56 ± 0.41	94.56 ± 0.41	94.56 ± 0.41
		DistBoost.F	88.63 ± 0.78	87.80 ± 1.53	91.53 ± 1.08	89.81 ± 1.83	87.01 ± 1.21	36.70 ± 6.89
		PreWeak.F	**93.83 ± 0.80**	93.48 ± 0.87	91.49 ± 1.17	92.62 ± 1.50	**94.42 ± 0.97**	46.16 ± 15.30
		AdaBoost.F	93.21 ± 0.80	**93.94 ± 0.54**	**93.88 ± 0.26**	**92.93 ± 0.99**	89.32 ± 2.00	**46.21 ± 3.87**
	vowel	Samme	86.16 ± 2.19	86.16 ± 2.19	86.16 ± 2.19	86.16 ± 2.19	86.16 ± 2.19	86.16 ± 2.19
		DistBoost.F	74.14 ± 4.25	70.51 ± 1.70	70.71 ± 2.79	68.89 ± 4.75	56.46 ± 8.12	27.27 ± 9.77
		PreWeak.F	77.88 ± 1.49	**80.71 ± 1.49**	**78.18 ± 2.12**	**77.78 ± 2.05**	**71.62 ± 5.81**	26.06 ± 10.75
		AdaBoost.F	**79.80 ± 1.47**	80.30 ± 2.72	77.27 ± 2.88	75.15 ± 4.78	66.67 ± 6.37	**31.21 ± 6.41**
	letter	Samme	75.33 ± 0.52	75.33 ± 0.52	75.33 ± 0.52	75.33 ± 0.52	75.33 ± 0.52	75.33 ± 0.52
		DistBoost.F	62.37 ± 2.61	61.26 ± 1.79	55.75 ± 0.98	61.11 ± 1.59	46.57 ± 8.63	21.12 ± 1.78
		PreWeak.F	**71.46 ± 1.85**	**71.93 ± 1.61**	**70.21 ± 1.50**	**69.90 ± 0.45**	64.34 ± 3.77	45.71 ± 3.92
		AdaBoost.F	68.32 ± 1.63	69.88 ± 0.78	66.58 ± 1.37	68.20 ± 0.93	**64.98 ± 4.04**	**54.96 ± 3.12**
	Avg. rank	Samme	-	-	-	-	-	-
		PreWeak.F	1.6	**1.5**	**1.5**	**1.143**	**1.333**	**1.5**
		DistBoost.F	2.9	2.9	2.9	3.0	2.429	2.9
		AdaBoost.F	**1.5**	1.6	1.6	1.875	2.000	1.6

TABLE II: Summary of F1 scores at $T = 300$. For each dataset/non-iidness type combination the federated method with the best F1 score is highlighted in bold. All F1 figures are multiplied by 100 for the sake of readability.

skew case are worse than those in the pathological skew: the corresponding points ($\otimes$ symbols) appear upper (w.r.t. ⬟) because the federation does not perform well in this particular case.

Table II provides all the average F1 scores ($\pm$ standard deviation) for all methods, datasets, and skewness. Overall, the performance of `PreWeak.F` and `AdaBoost.F` are significantly better than `DistBoost.F`. We can observe that, in general, the federation tends to achieve F1 scores very close to the centralized SAMME on datasets with few labels (e.g., 2 and 3), even in non-iid settings. Clearly, as the number of classes increases, the prior shift scenario becomes more and more challenging. The Labels Quantity skew is the most demanding setting because each client only has two labels. Thus, their weak classifiers are not good enough to be boosted effectively.

As it is evident from the average rank (bottom of Table II), `PreWeak.F` is the best performing algorithm; still, `AdaBoost.F` seems to achieve comparable results in most cases, especially in the iid (i.e., uniform) setting. However, in some experiments, `PreWeak.F` showed an inconsistent behaviour, as shown in the second row and second column of Figure 2. We conjecture that this is a form of overfitting due to the strong drive of `PreWeak.F` in reducing the training error, which is itself a consequence of the over-specialization of the locally-learned classifiers. In fact, sometimes (e.g., in the `adult` case), we observed that the F1 score of `PreWeak.F` is much higher on the training set (plot not shown due to space reasons) than on the test set, showing apparent symptoms of overfitting. In this case, however (i.e., on `kr-vs-kp`), this was not the case as the bad curve observed in the plot is also observable on the training set. Nonetheless, we argue that it is still overfitting but of a more subtle kind: the weak classifiers overfit the local data but they continue to do reasonably well until they are

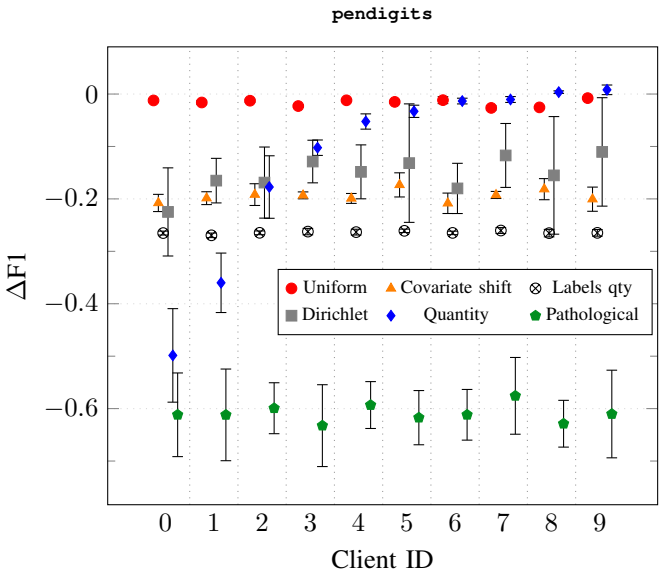

Fig. 1: Difference in F1 score (ΔF1) between the local run of SAMME and the best results with the federation. Results are reported for each clients and non-iidness. The scores are the average (± standard deviation) over 5 runs.

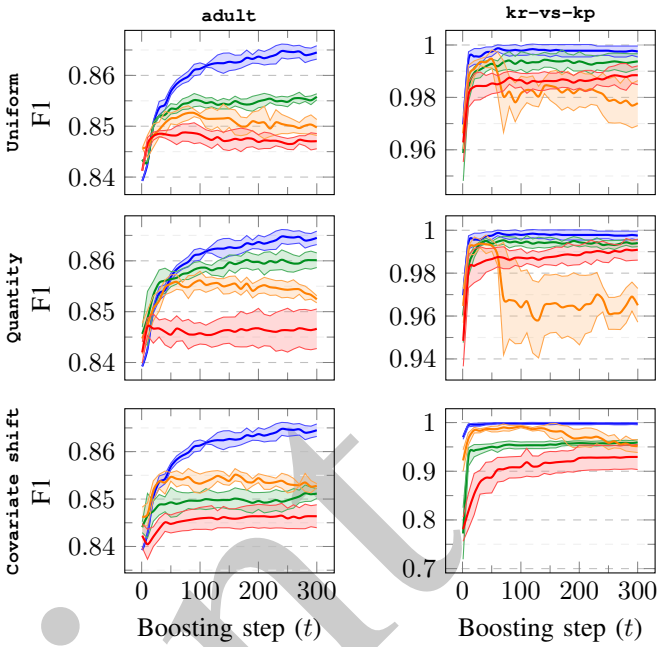

Fig. 2: Average F1-score curves on two binary classification datasets for all data distributions with $t \in [1, 300]$. The shady area is the standard deviation. Each colored curve represents a method: `Samme` (blue), `AdaBoost.F` (green), `PreWeak.F` (orange) and `DistBoost.F` (red).

combined in a sufficient number. Then the adaptation forced by AdaBoost actually hurts the ensemble. This behavior can happen with all the algorithms we presented. For instance, something similar is happening with `DistBoost.F` as shown in Figure 3 row 1, column 2, and can also be observed for `AdaBoost.F` (not reported here). However, we noticed this phenomenon more frequently with `PreWeak.F`.

In Figure 3 we report the results on two multi-class datasets, namely the dataset with the highest number of labels in our pool, i.e., `letter`, and (sat). Some of the same considerations made for the case of binary datasets holds here as well. We still see `DistBoost.F` not performing as well as the other approaches. The other two approaches have similar results with a slight edge in favor of `PreWeak.F`. On `letter` (left column) `PreWeak.F` and `AdaBoost.F` have very similar performances in the first 5 skewness configurations and a slight edge of `AdaBoost.F` in the last one. On `sat`, on the other hand, `PreWeak.F` is often the leading algorithm even outperforming the centralized SAMME implementation as shown in the top right panel. In the rest of the right panels `AdaBoost.F` does not perform as well as `PreWeak.F`, but usually still better than `DistBoost.F`.

Overall, we believe that the evidence presented here is enough to conclude that the approach is beneficial and that `DistBoost.F` is not performing as well as the other two algorithms. There is evidence, albeit not conclusive, that `PreWeak.F` outperforms `AdaBoost.F` in terms of performances and that `PreWeak.F` might suffer more than `AdaBoost.F` from overfitting problems.

V. Conclusions

The possibility of applying federated learning beyond gradient-based methods may broaden the adaptation of this methodology. In this paper, we exploit ideas from distributed boosting literature to propose three algorithms `DistBoost.F`, `PreWeak.F`, and `AdaBoost.F`, which allow, for the first time ever, the federation of parties without putting constraints on the type of models learned in the clients. Indeed, to the best of our knowledge, our proposal is also the first to allow each client to choose a different local model.

Our experiments show that the federation works. The generalization error of the federation is driven down by the three algorithms and, except in trivial cases, the federated model largely outperforms the models that could have been learned locally. Experiments also show that non-iid data distributions can harm the quality of the federated model. Specifically, when an extreme skew on the labels is present, the federation might suffer, especially when the problem is multi-class and the number of possible labels is large. We leave as future work a comparison between our approach and traditional (gradient-based) federated algorithms. The comparison would also allow us to assess how much the problems we observed in some non-iid settings are specific to our methodology.

Among the algorithms we proposed, `PreWeak.F` has the best performance in terms of F1 and should then be preferred in many cases. It is worth mentioning, however, that `AdaBoost.F` has some characteristics that might prove beneficial in some scenarios. In fact, in our experience, `PreWeak.F`

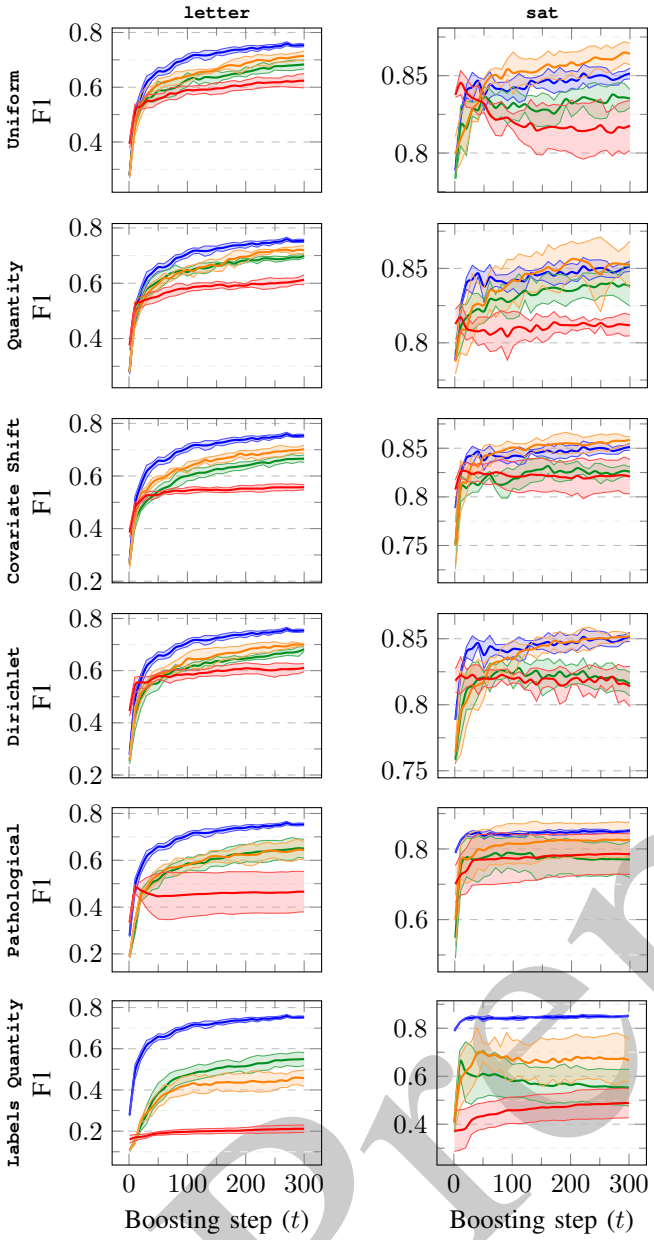

Fig. 3: Average F1-score curves on two multi-class datasets for all data distributions with $t \in [1, 300]$. The shady area is the standard deviation. Each colored curve represents a method: `Samme` (blue), `AdaBoost.F` (green), `PreWeak.F` (orange) and `DistBoost.F` (red).

seems less "stable" than `AdaBoost.F`. Also, `AdaBoost.F` is more flexible for what it concerns the number of learned weak classifiers, and it would then be easier and quicker to validate. For instance, one can early stop a validation round when using `AdaBoost.F`, thus saving computational time. In contrast, early stopping `PreWeak.F` is less beneficial since each client must run the complete AdaBoost algorithm independently.

This work opens the doors to many possible future directions. We aim to perform an in-depth analysis of these algorithms' security and privacy aspects in our future work. As mentioned, we would like to compare their behavior against gradient-based alternatives. Finally, we plan to deploy them in an actual federated environment (i.e., not simulated) to assess their performances from the point of view of memory, network, and CPU requirements.

References

[1] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] McMahan et al., "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[3] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[4] Y. Freund and R. E. Schapire, "Game theory, on-line prediction and boosting," in *Proceedings of the ninth annual conference on Computational learning theory*, pp. 325–332, 1996.

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[6] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[7] A. Lazarevic and Z. Obradovic, "Boosting algorithms for parallel and distributed learning," *Distributed and Parallel Databases*, vol. 11, pp. 203–229, Mar 2002.

[8] J. Cooper and L. Reyzin, "Improved algorithms for distributed boosting," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 806–813, 2017.

[9] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[10] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[11] E. B. Kong and T. G. Dietterich, "Error-correcting output coding corrects bias and variance," in *Machine learning proceedings 1995*, pp. 313–321, Elsevier, 1995.

[12] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.

[13] Y. Liu, Y. Liu, Z. Liu, J. Zhang, C. Meng, and Y. Zheng, "Federated forest," 2019.

[14] L. A. C. de Souza, G. Antonio F. Rebello, G. F. Camilo, L. C. B. Guimarães, and O. C. M. B. Duarte, "Dfedforest: Decentralized federated forest," in *2020 IEEE International Conference on Blockchain (Blockchain)*, pp. 90–97, 2020.

[15] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *Proc. VLDB Endow.*, vol. 13, pp. 2090–2103, jul 2020.

[16] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[17] M. Polato, A. Gallinaro, and F. Aiolli, "Privacy-preserving kernel computation for vertically partitioned data," in *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2021.

[18] S.-T. Chen, M.-F. Balcan, and D. H. Chau, "Communication efficient distributed agnostic boosting," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (A. Gretton and C. C. Robert, eds.), vol. 51 of *Proceedings of Machine Learning Research*, (Cadiz, Spain), pp. 1299–1307, PMLR, 09–11 May 2016.

[19] F. Wang, J. Ou, and H. Lv, "Gradient boosting forest: a two-stage ensemble method enabling federated learning of gbdts.," in *Internation Conference On Neural Information Processing (ICONIP)* (T. Mantoro, M. Lee, M. Ayu, K. Wong, and A. Hidayanto, eds.), vol. Lecture Notes in Computer Science, vol 13109, Springer, Cham., 2021.

[20] Y. Liu, Z. Ma, X. Liu, S. Ma, S. Nepal, R. H. Deng, and K. Ren, "Boosting privately: Federated extreme gradient boosting for mobile crowdsensing," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1–11, 2020.

[21] Y. Liu, Z. Fan, X. Song, and R. Shibasaki, "Fedvoting: A cross-silo boosting tree construction method for privacy-preserving long-term human mobility prediction," *Sensors*, vol. 21, no. 24, 2021.
[22] K. Cheng, T. Fan, Y. Jin, Y. Liu, T. Chen, D. Papadopoulos, and Q. Yang, "Secureboost: A lossless federated learning framework," *IEEE Intelligent Systems*, vol. 36, no. 6, pp. 87–98, 2021.
[23] W. Fang, D. Zhao, J. Tan, C. Chen, C. Yu, L. Wang, L. Wang, J. Zhou, and B. Zhang, *Large-Scale Secure XGB for Vertical Federated Learning*, pp. 443–452. New York, NY, USA: Association for Computing Machinery, 2021.
[24] F. Fu, Y. Shao, L. Yu, J. Jiang, H. Xue, Y. Tao, and B. Cui, *VF2Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning*, pp. 563–576. New York, NY, USA: Association for Computing Machinery, 2021.
[25] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, pp. 1–207, Dec. 2019.
[26] Kairouz et al., "Advances and Open Problems in Federated Learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
[27] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.
[28] D. Dua and C. Graff, "UCI machine learning repository," 2017.
[29] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," *CoRR*, vol. abs/2102.02079, 2021.