Machine Learning and Optimization Laboratory

Incentivized, Privacy-preserving, and Personalized Distributed Machine Learning Framework for Medical Data

Semester project

by Felix Grimberg

Mary-Anne Hartley
Supervisor

Martin Jaggi
Co-supervisor

Sai Praneeth Reddy Karimireddy
Co-supervisor

Andres Colubri
Co-supervisor

16/06/2020

Abstract

Contents

1 Aim

This project is part of a larger endeavour to create a platform for medical professionals around the globe, and especially in low-resource settings, to train machine learning (ML) models collaboratively. Main requirements to achieve this goal are identified and separated into necessary features and performance features.

1.1 Necessary features

Data privacy. First and foremost, the proposed platform must safeguard each participant from the leakage and re-identification of individual patients stored within their data set. This implies maintaining data at its original location and limiting transfers, which motivates the use of so-called cross-silo federated learning (FL) techniques [9].

Incentives and intellectual property. Secondly, the platform must present incentives for the collection of high-quality health data and protect the intellectual property of their owner.

Resilience. Moreover, it must be designed to work with incomplete and non-identically distributed data, as well as with arbitrary connection/disconnection patterns from participants.

Personalization. Finally, it is important that the selection of training data, features, and model architecture, is interpretable to the users (i.e., medical professionals) and subjected to their judgement. To this end, users must be given the tools to ensure that the trained ML model is adapted to the particularities of their local population.

1.2 Performance features

Data analysis. For better model personalization, the proposed platform should facilitate the selection of appropriate training data, e.g., by providing and visualizing measures of similarity between data sets.

Feedback. Additionally, it should give its users feedback on the quality of their data collection. For instance, it could point out physiological or epidemiological anomalies (e.g., breathing rates being consistently reported higher than usual). Likewise, it could suggest which additional features should be recorded to efficiently maximize future model performance and thus increase their collaborative scope.

Environmental impact. The framework should also be optimised to quantify, minimise and possibly compensate for the environmental impact associated with training each ML model.

Communication efficiency. The communication effort exerted by each participating device should be kept as low as possible, to facilitate participation in low-infrastructure settings. This simultaneously serves to reduce the environmental impact of each training run.

Robustness. Finally, the design should be robust to a small fraction of non-collaborative participants. Different robustness models exist for various types of non-collaboration, ranging from the inadvertent provision and use of false data to fully adversarial behaviour.

2 Objectives

Within the scope of this project, our main focus will be on developing and evaluating methods for model personalization. Throughout the project, we will be mindful of the other requirements identified in Section 1 and seek methods that are compatible with, or even beneficial for, as many of them as possible. We will strive to achieve these objectives in a fully reproducible manner, while maintaining the confidentiality of the provided data sets.

2.1 Landscape analysis

We will begin by researching existing methods for privacy-preserving distributed machine learning and model personalization. During this landscape analysis, we will also explore available tools for reproducible data science, as well as existing incentive schemes, and investigate how these tools can be leveraged to efficiently realise the other objectives listed herein. Our findings will be documented in Section 3 of this report.

2.2 Model personalization and data selection

Often, each agent in a FL setting collects data on a different population. For instance, a hospital in Freetown and an Ebola treatment center in rural Sierra Leone would see fairly different patients in general, which would again differ from the patients admitted in Ebola treatment facilities in the Democratic Republic of the Congo. While it is also highly useful to make global inference across populations, clinicians in the Freetown hospital may be more interested in being able to accurately predict outcomes for new patients admitted *to their hospital*, than in predicting outcomes for new patients admitted *to any facility in general.* They would still need to use each other's data collaboratively, simply because the amount of data collected by each user is limited.

We will propose a variety of novel ways to help users in optimising the selected model for their local population. Some of the proposed methods will involve ranking the data sets of other users, exploiting various notions of similarity to the local data set. Novel ideas that we will pursue are:

- **Ndoye:** Computing a (potentially adapted) Ndoye factor for each available user before training [15].
- **Influence:** Using a subset of the local data set as a test set, and then selecting other users according to how much they contribute to reducing the test loss. As an additional advantage, this would guarantee that the evaluation metrics of the resulting model measure its ability to make predictions on the local population (as opposed to the global population or some aggregated subset thereof).
- **Gradients:** Training a global model, where the mini-batch gradients computed by each user are used to define a notion of similarity. This concept is based on the idea that similar data sets will produce similar gradients in expectation over a mini-batch SGD step.

These ideas will be evaluated with respect to the following criteria:

- **User focus:** Freedom of choice left to the users and interpretability of the information presented to them.
- **Resilience:** Compatibility with the distributed machine learning setting, resilience to incomplete data and to unreliably connected users.
- **Incentives:** Potential to provide incentives and/or protect intellectual property.
- **Feedback:** Potential to generate feedback on data quality and feature collection for the user. This does not include feedback on the interoperability of data sets, as this can be done without any data selection method.
- **Privacy:** Compatibility with existing methods that ensure data privacy, such as secure multi-party computation[1] (MPC).
- **Byzantine robustness:** Compatibility with existing methods that ensure robustness against byzantine participants. By their purpose, all effective data selection methods are robust against the mere poisoning of other users' data and highly susceptible to the poisoning of the requesting user's own data.
- **Complexity:** Computation and communication requirements.

To that effect, ideas will be formalized into concrete methods. The methods and their evaluation with respect to the mentioned criteria will constitute Section 4.

[1] Put simply, the term *Secure multi-party computation* refers to a range of techniques that make it possible for two or more parties to jointly evaluate a function (e.g., `sum` for 3 or more parties) without revealing their individual inputs, cf. section 4.2.1 of [9].

2.3 Application case study

Next, we will implement these methods and apply them to a medical data set collected on 577 patients in Sierra Leone during the 2013 - 2015 West African epidemic of Ebola Virus Disease (EVD) [6]. When a larger or more complete data set is needed, we will use the publicly available Titanic dataset [11]. Finally, an application will be made to access the Ebola Data Platform (EDP) assembled by the Infectious Diseases Data Observatory [4]. The distributed ML problem will be simulated on a single machine. We will split the available data across simulated devices in a variety of ways to emulate unequal data set size, unequal distribution of features, unequal distribution of labels, unequal distribution of missing entries, and varying degrees of inequality between data sets.

We will evaluate how well each method performs for each split by using it to train and evaluate a prediction model. Two baseline methods will also be evaluated, where the model is trained once using only the local data set, and once using all available data. Evaluation metrics, such as root mean squared error or classification accuracy, will be computed on the local test set for the trained models. We will visualize the evolution of these metrics during the learning process, per split for the various methods.

3 Landscape analysis

3.1 Privacy in ML

A very comprehensive overview of the FL problem, including state-of-the-art methods and open research areas of data privacy, resilience, personalization, and robustness, is given by Kairouz et al. [9]. The main tools used to train ML models with data privacy guarantees are MPC[2], homomorphic encryption[3] (HE), and differential privacy[4] (DP). These tools are implemented in the open-source deep learning framework `PySyft`, which exposes familiar deep learning APIs like `PyTorch` and `TensorFlow` [19, 16].

A different approach is taken in [14]: Here, methods and metrics for privacy-preserving ML and knowledge discovery from data (KDD) are compared and contrasted for each stage of the data lifecycle: data collection, data publication and data output (e.g., in the form of a singular prediction). Strengths and drawbacks of existing DP, MPC, HE, randomization, and k-anonymity tools are discussed from a practical perspective.

As exposed by Kairouz et al., authors often address specific issues individually, such as data privacy *or* robustness [9]. For instance, `PySyft` does not (at the date of publishing) provide robustness against malicious users [19]. Nevertheless, a privacy-preserving *and* byzantine-robust[5] aggregation scheme is proposed in [7]. It relies on securely computing the pairwise distance between the gradients of each client at each round, and using them to select a byzantine-robust subset of clients at each round (e.g. via multi-KRUM [1]).

3.2 Model personalization

Classical ML algorithms often assume that all samples used at training, testing, and prediction time are independent and identically distributed (IID). In the FL setting, however, this assumption can be violated for various reasons detailed in section 3.1 of [9]. One of these reasons can be that each of the agents participating in the FL task collects data from a different distribution, such as in the example described in Section 2.2 of this report. In such cases, model performance can be improved by applying one or several methods, to which we shall refer as *personalization*. Section 3.3 of [9] outlines the following such personalization methods:

Featurization : If samples coming from distinct individuals can be assumed to follow different distributions based on certain personal context information, a natural approach is to include this context information as features in the ML model.

[2] For MPC, see footnote in Section 2.2.

[3] Homomorphic encryption consists of encrypting inputs in such a way that certain mathematical operations can be performed on cyphertext (i.e., on the encrypted version of the inputs) to produce the cyphertext of the correct outputs.

[4] Differential privacy aims to mask the contribution of any individual user to the ML model by adding a small amount of noise, in order to prevent or limit information disclosure. It is discussed in section 4.2.2 of [9].

[5] Byzantine-robustness is achieved if, given enough participating parties with enough data, the training process converges to a good model even when a given proportion (typically up to, but not including, 50%) of participants exhibit arbitrarily malicious behaviour. For instance, these so-called *byzantine participants* can make up arbitrary values whenever they communicate with the server or with other participants.

Multi-task learning : Alternatively, each agent's local problem can be viewed as a separate learning task. If the number of available samples per agent is too low, agents can also be clustered to define one task per cluster.

Local fine-tuning : This method consists of training one global model through FL, broadcasting it, and subsequently letting each agent personalize the model by additional training on their local data set.

Learning-to-Learn (LTL): Also called *Meta-learning*, LTL consists of learning a learning algorithm by sampling from a meta-distribution of ML tasks.

Model-agnostic Meta-learning (MAML): The goal of MAML is to learn a global model specifically as a good starting point for local fine-tuning. This contrasts the approach of locally fine-tuning a global model that was optimized exclusively for its accuracy, potentially at the expense of its "fine-tuneability".

We see that personalization can be achieved by clustering the agents and therefore selecting a subset of the global data set to train on. Nevertheless, it is not to be confused with *training data selection*, which refers to the task of selecting good samples from one large training set whose samples exhibit various levels of noise and relevancy [12].
A learning theoretic formulation of the personalized FL setting, along with three personalization approaches, is presented in [13]. The first presented method, *hypothesis-based user clustering*, is a form of multi-task learning where an appropriate agent clustering is found *during* the model training process. Next up, *data interpolation* aims to reach a good compromise between generalization properties and specificity of the resulting model, by minimizing the loss on a weighted combination of local data and global data. Finally, *model interpolation* is presented as a MAML-like method for the joint optimization of the global model, a global-to-local proportion for each agent, and the local model for each agent. In model interpolation, the purpose of the local models is really to correct wrong predictions of the global model, rather than to make reliable predictions on their own.

3.3 Incentives

Successful federated learning applications depend on the data collection efforts and the subsequent provision of high-quality data by all participating agents.

1. Overall problem
2. What people have done to solve this problem → literature

- [9]: "Incentive mechanisms In addition to developing new algorithmic techniques for FL, incentive mechanism design for honest participation is an important practical research question. This need may arise in cross-device settings (e.g. [225, 224]), but is particularly relevant in the cross-silo setting, where participants may also be business competitors. Related objectives include how to divide earnings generated by the federated learning model among contributing data owners in order to sustain long-term participation, and also how to link the incentives with decisions on defending against adversarial data owners to enhance system security, optimizing the participation of data owners to enhance system efficiency."
- [18]: Designs a scheme for the efficient approximation of the influence of each user's provided data (i.e., how much their data has contributed to minimising the loss function), which could serve as a basis for rewarding users. This requires that the payer has a private test set on which to compute the loss / the influence. Under this assumption, their scheme incentivises truthful data collection and reporting.

3.4 Reproducible data science

A variety of platforms like `Google Colaboratory`[6] and `Renku`[7] have been developed to enable reproducible data science by leveraging version control, virtual environments, and containerization and providing access

[6] `Google Colaboratory`: `https://colab.research.google.com/`

[7] `Renku`: `https://datascience.ch/renku/`

to expensive hardware such as GPUs and TPUs[8]. Others, such as Amazon Web Services (`AWS`)[9] and Data Version Control (`DVC`)[10], provide comprehensive services for the entire ML pipeline, from model development and tuning to production. Virtual environments are a great tool to ensure the reproducibility of results obtained by executing `Python` programs. Unfortunately, they incur substantial storage space usage due to the duplicate installation of the entire `Python` installation and packages. More parsimonious dependency management tools, such as `watermark`, simply rely on printing software versions in an `IPython` cell [17].

4 Model personalization and data selection

4.1 Setting and Objective

We consider a network of agents i, each collecting samples $\mathcal{S}_i = \left\{\mathbf{x}_i^{(n)}, y_i^{(n)}\right\}_{n=1,\ldots,N_i}$ from an underlying distribution $\mathcal{D}_i$. One agent, agent 0, is called the user and wishes to perform an inference task, such as (regularized) linear or logistic regression, to gain knowledge about the distribution $\mathcal{D}_0$ from which they collect data. For instance, agent 0 could wish to predict the label y_0^{new} of a new sample after observing its features $\mathbf{x}_0^{new}$, using some kind of approximation of the conditional probability $p_0\left(y|\mathbf{x}\right)$.

However, agent 0 knows that the number $|\mathcal{S}_0|$ of samples they have collected is fairly small for the chosen inference task. Luckily, agent 0 believes that *some* of the other agents collect sufficiently interoperable[11] samples $\mathcal{S}_i$ from sufficiently similar underlying distributions $\mathcal{D}_i$, that the *true loss*[12] of their model on $\mathcal{D}_0$ could be reduced by including $\mathcal{S}_i$ in the training process. Unfortunately, agent 0 also knows that this is not the case for *all* agents. Simply including all agents' samples equally in the training process therefore would not help reduce the true loss of agent 0's model.

The task of *model personalization*, in a broader sense, is to give agent 0 a training algorithm which discriminates between the available agents in some way to minimize the true loss of the resulting model on $\mathcal{D}_0$.

One class of model personalization methods relies on *data selection*, followed by standard decentralized training on the selected subset of data. The task of data selection is thus to help agent 0 select a subset $\mathcal{U}_{train}$ of agents whose samples will be included in the training process. The optimal subset $\mathcal{U}_{train}^*$ is that which minimizes the true loss of the resulting model on $\mathcal{D}_0$.

Model personalization. Formally, we are given:

- A set of agents $i \in \mathcal{U} = \{0, 1, \ldots, N\}$
  - Each agent i has collected a set of samples (called *data set*): $\mathcal{S}_i = \left\{\mathbf{x}_i^{(n)}, y_i^{(n)}\right\}_{n=1,\ldots,N_i}$
  - Each agent i collects their samples from an (unknown) underlying distribution $\mathcal{D}_i$: $\left(\mathbf{x}_i^{(n)}, y_i^{(n)}\right) \overset{i.i.d.}{\sim} \mathcal{D}_i$
  - It is assumed that the label y_0 is not independent of the features $\mathbf{x}_0$ under $\mathcal{D}_0 : p_0\left(y|\mathbf{x}\right) \neq p_0\left(y\right)$
- A class of models $\mathcal{M}$ s.t. $f : \mathcal{X} \to \mathcal{Y}\ \forall f \in \mathcal{M}$. For instance, $\mathcal{M}$ could be the class of linear models where $f\left(\mathbf{x}\right) = \mathbf{w}\mathbf{x}$, $\mathbf{w} \in \mathbb{R}^D$.
- A loss function: $\ell\left(y, \hat{y}\right)$. For instance, the loss function could be the mean squared error (MSE).

[8] GPU (graphics processing unit) and TPU (tensor processing unit) are processors which leverage same instruction, multiple data (SIMD) parallelism. The training of most ML models is a heavily data-parallel task, allowing GPUs and especially the specially-designed TPUs to achieve considerable speed-ups and energy efficiency gains over training on CPUs.

[9] `AWS`: `https://aws.amazon.com/`

[10] `DVC`: `https://dvc.org/`

[11] Within the context of an inference task, two data sets are called *interoperable* if they collect all features chosen for said inference task in the same way. It is sometimes possible to make two data sets interoperable by constructing the required features from other, originally available features. For example, a data set that records the date of symptom onset and the date of admission can be made interoperable with a data set that records the number of days since symptom onset at admission (w.r.t. an inference task for which this number of days is a relevant feature). For a general definition see [3].

[12] The *true loss* of a model on a distribution is formally defined in Equation 1. It is a quantity that can be obtained only analytically, and only if the model and the distribution are perfectly known. In practice, it is approximated by the *test loss* on a test set consisting of samples drawn from the distribution. This approximation is good if the samples of the test set are "sufficiently" numerous, drawn "sufficiently" independently, and "sufficiently" independent from the samples of the training set.

We define the true loss $\mathcal{L}_{\mathcal{D}_0}(f)$ of a model $f \in \mathcal{M}$ on $\mathcal{D}_0$:

$$\mathcal{L}_{\mathcal{D}_0}(f) = \mathbb{E}_{\left(\mathbf{x}_0^{new}, y_0^{new}\right)\sim\mathcal{D}_0}\left[\ell\left(y^{new}, f\left(\mathbf{x}^{new}\right)\right)\right] \tag{1}$$

And we attempt to find a training algorithm $\mathcal{A}$ which, given the set $\mathcal{U}$ of agents i with their individual data sets $\mathcal{S}_i$, the class of models $\mathcal{M}$, and the loss function ℓ, will produce the model $f = \mathcal{A}(\mathcal{U}, \mathcal{M}, \ell) \in \mathcal{M}$ which minimizes the true loss $\mathcal{L}_{\mathcal{D}_0}(f)$ on $\mathcal{D}_0$ (Equation 2). This is opposed to standard federated learning methods $\mathcal{A}_{standard}^{FL}$, which consider that all non-malicious agents collect samples from the same underlying distribution. Applying such a method in the personalized FL setting would minimize a weighted sum of the agents' distributions $\mathcal{D}_i$, $i \in \mathcal{U}$ instead, where the weights are often either $\lambda_i = \frac{1}{|\mathcal{U}|}\forall i \in \mathcal{U}$ or $\lambda_i = \frac{|\mathcal{S}_i|}{\sum_{i\in\mathcal{U}}|\mathcal{S}_i|}$ (Equation 3).

Personalized FL minimization problem: $$find: \ \underset{\mathcal{A}}{\arg\min}\ \mathcal{L}_{\mathcal{D}_0}\left(\mathcal{A}\left(\mathcal{U}, \mathcal{M}, \ell\right)\right) \tag{2}$$

Standard FL result: $$\mathcal{A}_{standard}^{FL}\left(\mathcal{U}, \mathcal{M}, \ell\right) = \underset{f\in\mathcal{M}}{\arg\min} \sum_{i\in\mathcal{U}} \lambda_i \mathcal{L}_{\mathcal{D}_i}(f) \tag{3}$$

Data selection The data selection problem is a variant of the model personalization problem, where we are also given a training algorithm $\mathcal{A}$ which, given a training set $\mathcal{S}$ of samples, produces a model $f_{\mathcal{S}}$: $f_{\mathcal{S}} = \mathcal{A}(\mathcal{S})$, $\hat{y} = f_{\mathcal{S}}(\mathbf{x})$. For instance, this could be ridge regression with a regularization parameter selected among a certain number of candidates to minimize the validation error in 4-fold cross-validation.
We introduce the following notation:

- A subset $\mathcal{U}_{train}$ of agents: $\mathcal{U}_{train} \subseteq \mathcal{U}$
- The corresponding training set: $\mathcal{S}_{train} = \bigcup_{j\in\mathcal{U}_{train}} \mathcal{S}_j$

And we aim to find the subset $\mathcal{U}_{train}^*$ which will minimize the loss of the trained model $f_{\mathcal{S}_{train}} = \mathcal{A}(\mathcal{S}_{train})$ for agent 0:

$$\mathcal{U}_{train}^* = \underset{\mathcal{U}_{train}\subseteq\mathcal{U}}{\arg\min}\ \mathcal{L}_{\mathcal{D}_0}\left(f_{\mathcal{S}_{train}}\right)$$

Fictive example The agents could be individual hospitals dispersed across one or several regions. The aetiology of common, generic symptoms such as fever is highly dependent on geographic location, where the rural setting suffers more faecal-oral and vector borne diseases such as hepatitis A and malaria, while fevers in the urban setting tend to be related to respiratory disease. A medical professional (agent 0) might want to train a diagnostic model to help diagnose the patients admitted to their urban hospital. The number N_0 of samples collected at their hospital, however, is too limited to yield a very good model, so agent 0 considers using data from other hospitals. Knowing that rural populations suffer from fairly different problems than agent 0's urban patients, agent 0 pre-selects only other urban hospitals – in other words, agent 0 performs manual data selection based on prior medical knowledge. However, agent 0 suspects the presence of other confounding variables that could cause the samples collected by *some* of the other urban hospitals to negatively affect the true loss of the trained model on $\mathcal{D}_0$. In this example, the underlying distribution $\mathcal{D}_0$ describes the population of all possible patients of agent 0.

4.2 Summarizing evaluation of the proposed methods

In the following discussion of proposed methods, we shall assume without loss of generality that the data sets $\mathcal{S}_i$ collected by each agent i are perfectly interoperable. Indeed, a non-interoperable data set $\mathcal{S}_i^-$ can always be made technically interoperable by filling in all missing features with arbitrary values[13]. The resulting data set $\mathcal{S}_i^+$ no longer truly corresponds to the distribution $\mathcal{D}_i$, and is likely less useful[14] than it would be if it had directly been sampled from $\mathcal{D}_i$ in an interoperable manner. This loss of usefulness is a consequence of the lack of interoperability in the original data set $\mathcal{S}_i^-$ and can only partially be leviated by filling in the missing features in an informed way (rather than arbitrarily).

[13] In most cases, one can even fill in the missing features with reasonable, non-arbitrary values. This increases the usefulness of the resulting data set $\mathcal{S}_i^+$.

[14] In the context of the data selection problem, a data set $\mathcal{S}_i$ is more useful than another data set $\mathcal{S}_j$, if agent i is more likely to be included in $\mathcal{U}_{train}^*$ than agent j.

	Ndoye	Influence	Gradients
User focus	o		-
Resilience	+		+
Incentives	+		o
Feedback	+		-
Privacy	+		+
Byzantine	-		o
Complexity	+		o

Table 1: Summarizing evaluation of the model personalization methods outlined in the remainder of this section, along the metrics introduced in Section 2.2.

4.3 The adapted Ndoye factor

This method is subdivided into a data selection step, followed by a standard decentralized training task. It differs from the original Ndoye factor, in that the available data sets $\mathcal{S}_i$ are ranked by increasing $\mathcal{D}_i$-to-$\mathcal{D}_0$ transfer loss of the corresponding models $f_i = \mathcal{A}\left(\mathcal{S}_i\right)$:

$$\mathcal{L}_i^{transfer} = \mathcal{L}_{\mathcal{D}_0}\left(f_i\right) - \mathcal{L}_{\mathcal{D}_i}\left(f_i\right)$$

In practice, this transfer loss can only be approximated:

$$\hat{\mathcal{L}}_i^{transfer} = \mathcal{L}_{\mathcal{S}_0}\left(f_i\right) - \hat{\mathcal{L}}_{\mathcal{D}_i}^{CV}\left(f_i\right)$$

Above, $\mathcal{L}_{\mathcal{S}_0}\left(f_i\right)$ represents the average error of f_i on the finite dataset $\mathcal{S}_0$, while $\hat{\mathcal{L}}_{\mathcal{D}_i}^{CV}\left(f_i\right)$ corresponds to the approximation obtained by cross-validation. Agent 0 then uses this ranking to select a training set $\mathcal{S}_{train}$, e.g. by selecting $\mathcal{U}_{train}$ as the k agents with the lowest transfer loss or by selecting all agents whose transfer loss is below a given threshold.

It is clear why it is sensible to rank agents by increasing value of $\mathcal{L}_{\mathcal{D}_0}\left(f_i\right)$: If it is low, this most likely implies that $\mathcal{S}_i$ is well-suited as training data for the inference task specified by agent 0. Unfortunately, $\mathcal{L}_{\mathcal{D}_0}\left(f_i\right)$ depends not only on the distribution $\mathcal{D}_i$ (as well as $\mathcal{D}_0$ and $\mathcal{A}$), but also on the size $|\mathcal{S}_i|$ (i.e., the number of samples contained in $\mathcal{S}_i$). However, the size $|\mathcal{S}_i|$ no longer matters when the data sets of several users are combined into $\mathcal{S}_{train}$. Thus, the influence of $|\mathcal{S}_i|$ is removed by considering the transfer loss $\mathcal{L}_i^{transfer}$ instead of just the loss $\mathcal{L}_{\mathcal{D}_0}\left(f_i\right)$.

User focus. This method can be explained intuitively as evaluating how successfully an inference made on $\mathcal{S}_i$ can be transferred to $\mathcal{S}_0$. Thus, users can understand the ranking process, albeit without gaining any information as to *why* any particular data set was ranked higher than another. Since the ranking happens before any decentralized training, the user is free to either investigate this question by means of further data analysis before manually selecting $\mathcal{U}_{train}$, or select $\mathcal{U}_{train}$ automatically.

Resilience. This method judges each data set as a whole, thus incomplete data is not particularly problematic. Further, unreliably connected agents can be dealt with quite easily, as any existing, resilient decentralized training scheme can be used. The ranking only requires one message from agent 0 to each other agent, and back. Agent 0 can be given the option to select a subset $\mathcal{U}_{train}$ and start training before all other agents have responded.

Incentives. Agents can be rewarded, either proportionally to their rank, or simply based on whether they were included in $\mathcal{U}_{train}$. Finally, agent 0 can publish their selected $\mathcal{U}_{train}$, which would enable each other agent to check whether their contribution is sufficiently diluted to protect their intellectual property. Since this happens before the training process begins, agents can withdraw from any proposed inference tasks that would impinge too much on their intellectual property.

Feedback. If the model weights of f_i are revealed to agent 0, then they can be used to spot differences between $\mathcal{S}_0$ and $\mathcal{S}_i$, at the expense of training a model $f_0 = \mathcal{A}\left(\mathcal{S}_0\right)$. The other agents can receive extensive feedback based on their ranking, whether or not they were included in $\mathcal{U}_{train}$, and which model weights were particularly different from f_0.

Privacy. $\mathcal{L}_{\mathcal{S}_0}\left(f_i\right)$ can be computed via MPC without revealing $\mathcal{S}_0$ to agent i nor f_i to agent 0, albeit at a high communication cost.

Byzantine robustness. This method is not robust against byzantine agents. While a byzantine-robust federated or distributed training algorithm can be used, the user selection process relies on each agent to honestly and correctly report their cross-validation loss $\hat{\mathcal{L}}^{CV}_{\mathcal{D}_i}\left(f_i\right)$.

Complexity. In addition to the decentralized training process itself, this method merely requires each agent i to train one model f_i (entirely locally) and to send its weights, along with a singular loss value, back to agent 0. If MPC is used to hide the weights of f_i from agent 0, then the communication requirements increase drastically.

4.4 Influence-based model personalization

User focus.

Resilience.

Incentives.

Feedback.

Privacy.

Byzantine robustness.

Complexity.

4.5 Gradient-based model personalization

At the core of this method is a novel adaptation of federated training algorithms based on stochastic gradient descent (SGD). It is based on the secure (i.e., privacy-preserving) and byzantine-robust federated SGD-based ML framework proposed in [7]. Like said framework, this method relies on round-based gradient descent, and on securely computing distances between gradients through MPC at each round. The novelty lies in how these distances are used:
At each round, each agent i (including agent 0) computes a gradient:

$$\mathbf{g}_i = \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{S}_i}\left(f\right)$$

Above, $\mathbf{w}$ stands for all learned parameters of the current model f. The (normalized) distance between $\mathbf{g}_i$ and $\mathbf{g}_0$ is used as a metric for how different $\mathcal{D}_i$ is from $\mathcal{D}_0$:

$$d^{rel}_{i,0} = \frac{\|\mathbf{g}_i - \mathbf{g}_0\|}{\|\mathbf{g}_0\|} \in \mathbb{R}^+$$

Let us analyze what happens in a few examples, if $\mathbf{g}_i$ are indeed the full gradients (i.e., computed on the entire set $\mathcal{S}_i$), as opposed to stochastic gradients. Let us further assume that each training loss function $\mathcal{L}_{\mathcal{S}_i}\left(f\right)$ is convex in the model parameters $\mathbf{w}$.

Example 1 *Suppose the model has just been randomly initialized to f^0 and performs very suboptimally for both $\mathcal{S}_i$ and $\mathcal{S}_0$. Then, the normalized distance between the gradients will be very low. Formally:*

$$\begin{aligned}
& \min\left\{\mathcal{L}_{\mathcal{S}_0}\left(f^0\right), \mathcal{L}_{\mathcal{S}_i}\left(f^0\right)\right\} \gg \max\left\{\mathcal{L}_{\mathcal{S}_0}\left(f_{\mathcal{S}_i}\right), \mathcal{L}_{\mathcal{S}_i}\left(f_{\mathcal{S}_0}\right)\right\}, \quad f_{\mathcal{S}_i} = \underset{f\in\mathcal{M}}{\arg\min}\, \mathcal{L}_{\mathcal{S}_i}(f) \\
\Rightarrow \quad & \|\mathbf{g}_i - \mathbf{g}_0\| \ll \|\mathbf{g}_0\| \\
\Rightarrow \quad & d_{i,0}^{rel} \ll 1
\end{aligned}$$

Therefore, both agents i and 0 are fully included at this stage of the training process, because the distance $d_{i,0}^{rel}$ is close to 0.

However, as the training process progresses and the model gradually performs better, the distance between the gradients steadily increases. We can analyze the edge cases where f is either the global model or the local model:

Example 2 *Suppose f is the global model for 2 agents, 0 and i:*

$$\begin{aligned}
& f = \underset{f'\in\mathcal{M}}{\arg\min}\, \mathcal{L}_{\mathcal{S}_0\cup\mathcal{S}_i}(f'), \quad \mathcal{L}_{\mathcal{S}_0\cup\mathcal{S}_i}(f) = \frac{|\mathcal{S}_0|\mathcal{L}_{\mathcal{S}_0}(f) + |\mathcal{S}_i|\mathcal{L}_{\mathcal{S}_i}(f)}{|\mathcal{S}_0| + |\mathcal{S}_i|} \\
\Rightarrow \quad & \mathbf{0} = \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{S}_0\cup\mathcal{S}_i}(f) = \frac{|\mathcal{S}_i|}{|\mathcal{S}_0| + |\mathcal{S}_i|}\left(\frac{|\mathcal{S}_0|}{|\mathcal{S}_i|}\mathbf{g}_0 + \mathbf{g}_i\right) \\
\Rightarrow \quad & \mathbf{g}_i = -\frac{|\mathcal{S}_0|}{|\mathcal{S}_i|}\mathbf{g}_0 \\
\Rightarrow \quad & d_{i,0}^{rel} = \frac{\|\mathbf{g}_i - \mathbf{g}_0\|}{\|\mathbf{g}_0\|} = \frac{\left\|\left(-\frac{|\mathcal{S}_0|}{|\mathcal{S}_i|} - 1\right)\mathbf{g}_0\right\|}{\|\mathbf{g}_0\|} = 1 + \frac{|\mathcal{S}_0|}{|\mathcal{S}_i|}
\end{aligned}$$

In this case, whether (and to which degree) agent i should participate in the training depends on the sizes $|\mathcal{S}_0|$ and $|\mathcal{S}_i|$: Indeed, if $|\mathcal{S}_0| \ll |\mathcal{S}_i|$, then $d_{i,0}^{rel} \approx 1$, indicating that it would be useful to incorporate agent i further in the training process. This is sensible, considering that the much larger number of samples in $\mathcal{S}_i$ could help reduce the generalization error substantially. Inversely, if $|\mathcal{S}_0| \gg |\mathcal{S}_i|$, then it is not useful to include $\mathcal{S}_i$ in training and we are better off only using the (much more numerous) samples collected by agent 0. Correspondingly, this leads to $d_{i,0}^{rel} \gg 1$.

Example 3 *Now let us extend the previous edge case to more agents:*

$$\begin{aligned}
& f = \underset{f'\in\mathcal{M}}{\arg\min}\, \mathcal{L}_{global}(f'), \quad \mathcal{L}_{global}(f) = \frac{\sum_{i\in\mathcal{U}} |\mathcal{S}_i| \cdot \mathcal{L}_{\mathcal{S}_i}(f)}{\sum_{i\in\mathcal{U}} |\mathcal{S}_i|} \\
\Rightarrow \quad & \mathbf{0} = \nabla_{\mathbf{w}}\mathcal{L}_{global}(f) \\
\Rightarrow \quad & \exists i \in \mathcal{U} : \langle \mathbf{g}_i, \mathbf{g}_0 \rangle < 0 \\
\Rightarrow \quad & \exists i \in \mathcal{U} : \|\mathbf{g}_i - \mathbf{g}_0\| > \|\mathbf{g}_0\| \\
\Rightarrow \quad & \exists i \in \mathcal{U} : d_{i,0}^{rel} > 1
\end{aligned}$$

In this setting, 1 is indeed the best lower bound we can prove. To illustrate this, imagine a task with N agents whose gradients are $\mathbf{g}_{i,i>0} = -\frac{1}{N-1}\mathbf{g}_0$ at the global model f. Then $N \to \infty \Rightarrow \mathbf{g}_i \to \mathbf{0} \Rightarrow d_{i,0}^{rel} \to 1$. In that case, the low distance values suggest we should include them all in the training (in other words: stick close to the global model) because this greatly increases the number of available samples.

In general, in this example, relying on the distance measures implies moving towards the model best suited for a weighted subset of data sets (those with $d_{i,0}^{rel} \approx 1$) and neglecting data sets that are in stark disagreement with $\mathcal{S}_0$ (those with $d_{i,0}^{rel} \gg 1$).

Example 4 *Suppose now that f is the local model of agent 0:*

$$
\begin{aligned}
& & f = \underset{f' \in \mathcal{M}}{\arg\min}\ \mathcal{L}_{\mathcal{S}_0}\left(f'\right) \\
\Rightarrow & & \mathbf{g}_0 = \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{S}_0}\left(f\right) = \mathbf{0} \\
\Rightarrow & & \forall i \in \mathcal{U} : d_{i,0}^{rel} \ \textit{undefined}\ (+\infty)
\end{aligned}
$$

Under the stated assumptions of convexity and full gradient descent, all distances $d_{i,0}^{rel}$ grow without an upper bound as the trained model approaches the local model. In other words, when we move too close to the local model, all data sets start to appear very different from $\mathcal{S}_0$. Consequently, using any decreasing function of $d_{i,0}^{rel}$ as a similarity metric creates the danger of converging to the local model. One possible strategy to prevent this is to stop training before convergence. Intuitively, training should not be stopped as long as the gradients fit into a D-dimensional cone[15].

Weight Erosion. We propose an adapted SGD scheme in Algorithm 1, which accounts for the limitations highlighted in these examples and can be implemented securely using the two-server framework presented in [7]. Each agent i is initially given a weight of $\alpha_i^0 = 1$. At each round, they compute a minibatch gradient[16] and their weight α_i is decreased by a small amount $\Delta\alpha_i$ that depends on $d_{i,0}^{rel}$. Thus, all agents' weights decrease over time until they either reach 0 or the maximum number of rounds is reached. Nevertheless, the speed at which α_i decreases depends on how similar $\mathcal{S}_i$ is to $\mathcal{S}_0$. Unlike the adapted Ndoye factor, the result of the Weight Erosion scheme depends on how the model is initialized in round 0. However, the round-based approach gives it more resilience to randomly appearing or disappearing agents.

The batch size is fixed across all agents, such that all samples are treated equally in the first epoch. However, as the number of rounds increases, samples from smaller data sets will be seen more often than samples from larger data sets, because each agent uses the same number of samples per round. This leads to an over-representation of samples from smaller data sets. To counteract this, $\Delta\alpha$ is made to depend also on the average number of times each sample in $\mathcal{S}_i$ has been used. The same issue could alternatively be compensated by assigning another, independent and fixed weight to each agent.

Algorithm 1 relies on four hyperparameters: The number of rounds r_{max}, the batch size b, the distance penalty factor p_d, and the size penalty factor p_s. The former two are already inherent to SGD, while the penalty factors are introduced to calculate $\Delta\alpha$. One natural choice for the newly introduced parameters would be $p_s = 0$ and $p_d = 0.05$. With these parameters, for example, a gradient $\mathbf{g}_i$ of the same length as $\mathbf{g}_0$ would lead to a decrease by $\Delta\alpha = \frac{1}{24}$ if it is at a 45° angle with $\mathbf{g}_0$, or by $\Delta\alpha = 0.1$ if the gradients are at a 180° angle. p_s would have

[15] If all gradients fit into a cone, then it is possible to improve the loss on all data sets by taking a step in the opposite direction of the axis of the cone.

[16] Alternatively, the agents can compute local SGD updates, which are used in the same way.

to be chosen differently if some agents have much larger data sets than others.

Algorithm 1: Weight Erosion

```
Data: A set of agents $i \in \mathcal{U}$, with associated data sets $\mathcal{S}_i$, a model class $\mathcal{M}$, and a gradient-based machine
      learning algorithm $\mathcal{A}$.
Result: A personalized machine learning model $f$.
Set a number of rounds $r_{max}$, a batch size $b$, a distance penalty factor $p_d$, and a size penalty factor $p_s$ ;
Initialize $\alpha_i^0 \leftarrow 1 \forall i \in \mathcal{U}$ ;
Randomly initialize the model $f \in \mathcal{M}$ ;
for $r$ from 1 to $r_{max}$ do
    for $i \in \mathcal{U}$, starting with $i = 0$ do
        Select a batch of size $b$ from $\mathcal{S}_i$ and compute a gradient $\mathbf{g}_i$ ;
        Compute the distance $d_{i,0}^{rel}$ ;
        $\Delta\alpha \leftarrow \left(1 + p_s \left\lfloor \frac{(r-1)b}{|\mathcal{S}_i|} \right\rfloor\right) p_d d_{i,0}^{rel}$ ;
        $\alpha_i^r \leftarrow \max\left\{0, \alpha_i^{r-1} - \Delta\alpha\right\}$ ;
    end
    $\bar{\mathbf{g}} \leftarrow \frac{\sum_{i \in \mathcal{U}} \alpha_i^r \mathbf{g}_i}{\sum_{i \in \mathcal{U}} \alpha_i^r}$ ;
    Update the model $f$ based on $\bar{\mathbf{g}}$ ;
end
```

User focus. With this approach, the user does not learn why a certain subset of the available data sets were selected, even if they understand the selection process. They learn nothing beyond "These are the data sets for which $p^i(y|\mathbf{x})$ is similar to the local set". To make things worse, the selection is only revealed to them after the model has been trained.

Resilience. Like the algorithm proposed in [7], Algorithm 1 does not break if users appear or disappear between subsequent rounds. If this is expected, $\Delta\alpha$ can be modified by replacing r with the number of rounds in which agent i has actually participated. When a new agent appears after the first round, their weight should not be initialized to 1, but rather to the mean or median weight of all agents.

Incentives. Agents could be rewarded according to how many rounds they participated in, or according to the sum of their weights in each of these rounds. The examples give us an intuition that the result of Algorithm 1 lies somewhere between the global model and agent 0's local model. As long as the number and data set size of other users in $\mathcal{U}$ is sufficient to dilute each individual agent's contribution, the algorithm therefore does not impinge anyone's intellectual property. This condition on $\mathcal{U}$ can be verified by each agent before engaging in a round of training.

Feedback. Unfortunately, Algorithm 1 does not produce any information that lends itself to giving feedback on the user's data collection. The latter could be used

Privacy. A secure (i.e., privacy-preserving) and byzantine-robust SGD protocol based on a framework with two non-colluding servers is proposed in [7], which relies on secret sharing to compute and reveal the distances between gradients through MPC. Weight erosion can be seamlessly integrated into that protocol as an aggregation rule, instead of the byzantine-robust aggregation rule KRUM [1].

Byzantine robustness. Since weight erosion would *replace* the byzantine-robust aggregation rule in the setting of [7], the byzantine robustness property is lost. While byzantine inputs can negatively affect the training process, a byzantine agent i would likely see their weight α_i decline fast.

Complexity. This method relies on computing distances between gradients at each round. However, at each round, only $\mathcal{O}(|\mathcal{U}|)$ distances $d_{i,0}^{rel}$ are computed, while the byzantine-robust aggregation scheme in [7] computes $\mathcal{O}\left(|\mathcal{U}|^2\right)$ pairwise distances $\|\mathbf{g}_i - \mathbf{g}_j\|$ to select a byzantine-robust subset.

5 Application case study

We choose to implement the Weight Erosion scheme from Section 4.5. Why? In this section, ...

5.1 Implementation

The Weight Erosion scheme (Algorithm 1) is implemented on top of an early version of the `JAX` and `Haiku` based framework for FL simulations built by S. P. Karimireddy [10, 2, 8]. Changes were made to implement the Weight Erosion aggregation rule, to enable loading data sets from local files, as well as to report performance metrics of the trained model after each communication round. The just-in-time compilation API of `JAX` is used to speed up the computation of the distances $d_{i,0}^{rel}$.
Third-party reproducible data science platforms, such as those introduced in Section 3.4, are forgone to avoid copying the sensitive medical data sets introduced in Section 2.3 to third-party servers. Instead, all code for this project will be hosted on `GitHub` (available from within the `epfl-iglobalhealth` organization) and executed on the hardware provided by the Machine Learning and Optimization Laboratory (MLO). The repository is cited as a source [5]. The use of virtual environments is forgone in favor of `watermark` due to the high storage space usage discussed in Section 3.4 [17].

5.2 Case study

Predicting Ebola infection. While our request to access the Ebola Data Platform (cf. Section 2.3) has been granted, the data transfer could not be arranged on time for use within this semester project.
Thankfully, M. Hartley granted access to the raw dataset of medical data collected on 577 patients admitted at the GOAL-Mathaska Ebola Treatment Center in Port Loko, Sierra Leone, in 2014 and 2015 used in her 2017 publication [6]. This data set is used in a FL simulation, in an attempt to learn a prediction model for the Ebola diagnostic (EVD(+) vs EVD(-)) based on the features used in the triage score proposed by Hartley et al.: *Ebola contact history, days since first symptoms, conjunctivitis, diarrhoea, dysphagia, haemorrhage, fever >38° C*, and *myalgia* [6]. The raw dataset is pre-processed following the procedure outlined in [6], mainly to account for missing values. The exact pre-processing steps are documented on `GitHub` [5].

Splitting the data set across users. Due to the small size of the Ebola data set compared to typical ML data sets, the number of agents is kept minimal (3 agents). The data set is split among the three agents in the following ways:

`AGE_STRICT`: Samples are strictly segregated into three groups based on their age. For instance, agent 0: patients aged 0 - 20 years old / agent 1: 21 - 40 years old / agent 2: 41+ years old. This split should not introduce any label skew, since the age distribution is the same in the EVD(+) population as in the EVD(-) population. We expect a noticeable feature skew, however, since the prevalence of Malaria is strongly correlated with age. This feature skew should be sufficient to affect the conditional probability $p(y|\mathbf{x})$.

`AGE_SOME`: A subset of agents randomly share the totality of a given age group, while other agents contain other age groups exclusively. For instance, agents 0 and 1 randomly partition the patients aged 0 - 40 years old among themselves / agent 2 has all patients aged 41+ years old.

5.3 Results

The federated training of a prediction model by three agents is simulated on a single machine for the data sets and splits detailed in Section 5.2. Each simulation is repeated three times, such that each agent serves as user once. Every time, the user's data set is split into a test set and a training set of equal sizes, whereas 100% of the other agents' data sets are used as training sets. Three models are then trained:

`Global`:	A model trained on all agents' training sets without Weight Erosion.
`Weight Erosion`:	A model trained on all agents' training sets with Weight Erosion.
`Local`:	A model trained only on the user's training set.

At each communication round, each trained model's accuracy is measured on the user's test set and reported along with the weight α_i of each agent in the Weight Erosion scheme.

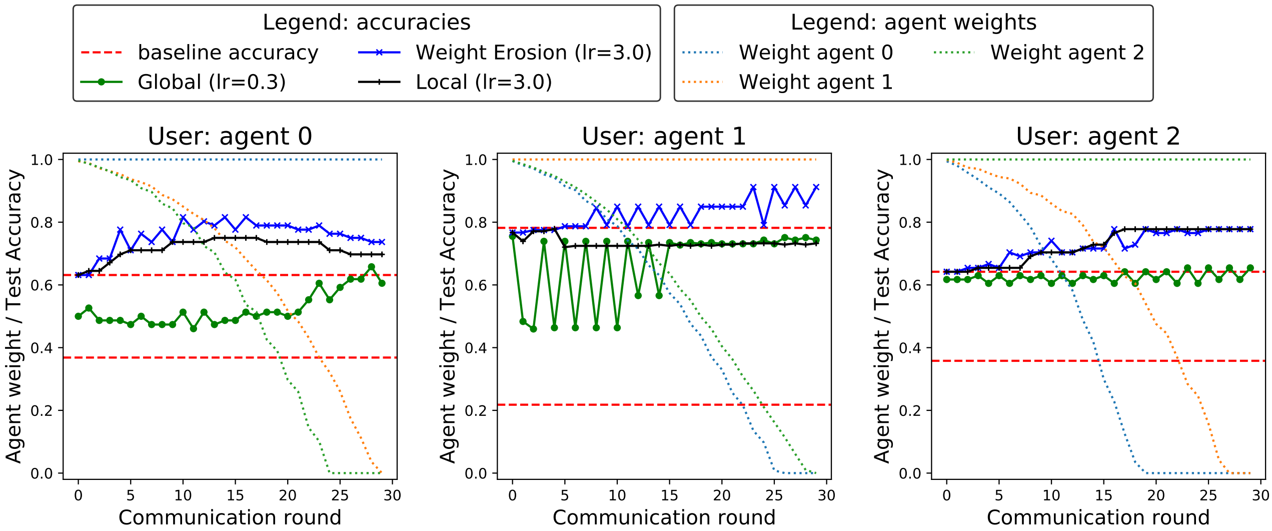

Figure 1: Simulation of federated training on the Ebola data set, split across 3 agents by age group:
agent 0: 0 - 20 years old / agent 1: 21 - 40 years old / agent 2: 41+ years old.
Full lines represent each model's accuracy on the user's test set. In red (dashed), the accuracy obtained by predicting always EVD(+) or always EVD(-). The pointed lines represent the weight α_i of each agent in the Weight Erosion scheme. The learning rates were tuned independently for each aggregation scheme (`Global`, `Weight Erosion`, `Local`), and they are displayed in the legend. One batch of 125 samples per round and agent (The user's training set contains only one such batch, the other agents' training sets contain two batches each.)
$p_d = 0.01$, $p_s = 0.2$, seed = 278

The lower of the two red dotted lines in Figure 1 corresponds to the prevalence of EVD(+) in the test set of the user. Contrary to our expectations, it reveals a considerable feature skew, at $p_1(y) < 0.25$ versus $p_0(y) \approx p_2(y) > 0.35$. This might help explain why the global model performs much less well than the local model. In this particular example, the Weight Erosion scheme seems to be able to outperform local training on only the user's data, albeit not always (Figure 1, right).

How fast the weight of each agent decreases depends on a number of influences:
Firstly, it depends on how similar the data sets $\mathcal{S}_i$ and $\mathcal{S}_0$ are with respect to the inference task at hand. This influence causes the weight of each agent to decrease at a different rate. In particular, how similar the data sets are, depends on the underlying distributions $\mathcal{D}_i$ and $\mathcal{D}_0$, as well as random chance if $|\mathcal{S}_i|$ and/or $|\mathcal{S}_0|$ are small.
Secondly, as discussed in Section 4.5, $d^{rel}_{i,0}$ also depends on how well or how poorly the trained model is currently performing.
And finally, the weight of all agents decreases more rapidly as the number of communication rounds increases, due to the influence of the size penalty factor p_s. The weight of each agent i decreases faster if $|\mathcal{S}_i|$ is smaller.
In Figure 2, $\mathcal{S}_1$ and $\mathcal{S}_0$ draw their samples from the same population (the age group 0 - 40 years old), thus we have $\mathcal{D}_0 = \mathcal{D}_1$. Disregarding the effect of random chance in the partitioning process, we would therefore expect the weight of agent 2 to decline noticeably faster than that of agent 1 (respectively agent 0), when agent 0 (respectively agent 1) is the user. We would also expect the weights of agents 0 and 1 to remain fairly similar throughout the simulation when agent 2 is the user. In other words, we would expect the dotted lines to be noticeably further apart (with the green dotted line below the other) in Figure 2 (left) and Figure 2 (center), than in Figure 2 (right).
On the contrary, we observe that the dotted lines are approximately the same distance apart in all three graphs of Figure 2, and that the green dotted line is above the orange one in Figure 2 (left). It could simply be that $\mathcal{D}_0 = \mathcal{D}_1 \approx \mathcal{D}_2$, which would explain this observation. This implies that we should reconsider our assumption that splitting the data set by patient age introduces a large enough feature skew to affect the conditional

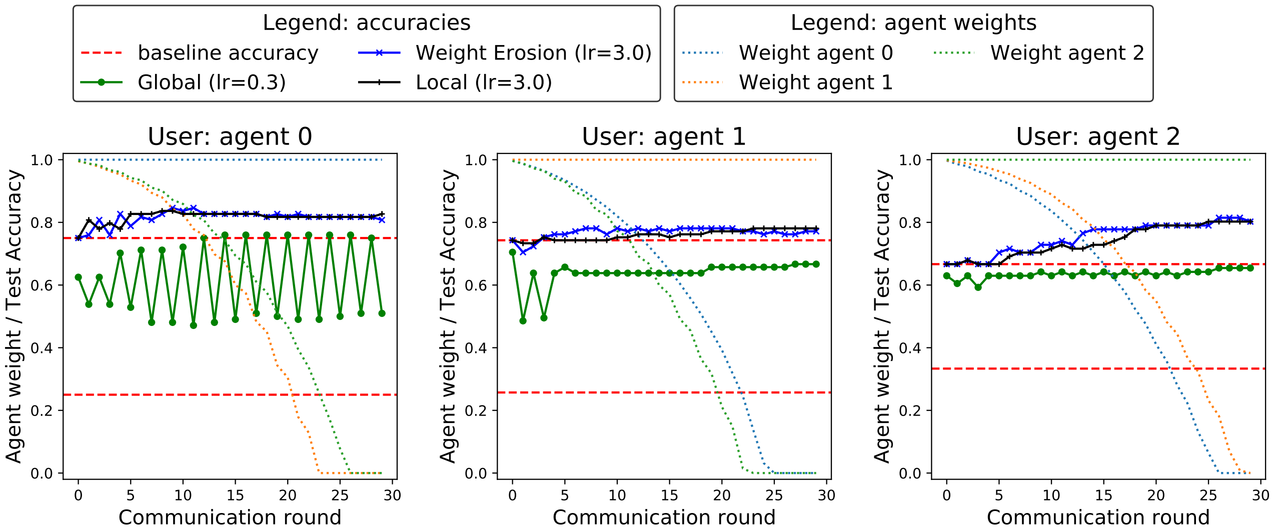

Figure 2: Simulation of federated training on the Ebola data set, split across 3 agents by age group:
agent 0 and agent 1: 0 - 40 years old / agent 2: 41+ years old.
Full lines represent each model's accuracy on the user's test set. In red (dashed), the accuracy obtained by predicting always EVD(+) or always EVD(-). The pointed lines represent the weight α_i of each agent in the Weight Erosion scheme. The learning rates are displayed in the legend. One batch of 125 samples per round and agent (The user's training set contains only one such batch, the other agents' training sets contain two batches each.)
$p_d = 0.01$, $p_s = 0.2$, seed $= 278$

probability $p(y|\mathbf{x})$. A different explanation would be that the distance $d_{i,0}^{rel}$ depends much more on how well the trained model is currently performing, than on how useful each agent's data set is in comparison with the other agents' sets.

6 Outlook

Shortcomings:

- Local model $f_0 = \underset{f \in \mathcal{M}}{\arg\min}\, \mathcal{L}_{\mathcal{S}_0}(f)$ is an attractor because $\mathbf{g}_0 = \mathbf{0} \Rightarrow d_{i,0}^{rel}$ arbitrarily large.
- Requires local data set $\mathcal{S}_0$ large enough to split into a meaningful test set and a meaningful training set.

Done, but not in the report:

- **Influence (add or remove)**:
  - Section 2.2
  - Section 3
  - Section 4
- **Write first paragraph of Section 3, Section 4, Section 5**
- **Write Conclusion, write Outlook as text**
- **Write abstract**
- **Check spelling *and grammar***
- Titanic dataset

- predict age (continuous variable)
- Manual hyperparameter tuning
- Discuss / evaluate methods from background along the same criteria and add them to the table

More to do:

- Automate hyper-parameter tuning and use cross-validation
- Split the data set in more different ways (**ESPECIALLY, NOT BY AGE**)
  - IID
  - Vary number of clients
  - By gender
  - By outcome (label skew)
  - skew $p\left(\mathbf{x}|y\right)$
  - skew $p\left(y|\mathbf{x}\right)$
- Test the gradient-based method on a different task
  - Different classification or prediction task on same dataset
  - Different model type with more parameters (on much larger dataset)
- Investigate a different Weight Erosion scheme where the over-representation of samples from smaller data sets is countered differently (cf. Section 4.5). E.g., α_i only decreases by $p_d \cdot d_{i,0}^{rel}$ and the gradients are weighted by $\alpha_i \left(1 - p_s \left\lfloor \frac{(r-1)b}{|\mathcal{S}_i|} \right\rfloor\right)$. But there might be a better way that truly does not over-represent smaller data sets.
- Implement other methods
- Refine gradient-based similarity some more, try it out on other problems
- Rigorously define similarity (e.g., based on learning theoretic notion of *discrepancy* [13]).
- Show summarizing metric (instead of evolution over training process) to compare different methods for various use cases (different data set splits, different tasks)
  - Best possible model obtained with the method (using cross-validation, else we get a random ranking)
  - Test accuracy after a certain number of SGD steps
  - Test accuracy within a certain communication traffic (total or per participant)
  - Test accuracy within a certain collaborative scope
- Evaluate the methods qualitatively by comparing which data sets are deemed similar to each other for any given task, and analyzing how these patterns arise.

References

[1] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. "Machine learning with adversaries: Byzantine tolerant gradient descent". In: *Advances in Neural Information Processing Systems*. 2017, pp. 119–129.

[2] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.55. 2018. URL: `http://github.com/google/jax`.

[3] *Definition: Interoperability*. URL: `http://interoperability-definition.info/en/` (visited on 06/01/2020).

[4] *Ebola Data Platform — Infectious Diseases Data Observatory*. URL: `https://www.iddo.org/ebola/data-sharing/accessing-data` (visited on 06/11/2020).

[5] Felix Hans Michel Grimberg and Martin Jaggi. *Semester project on private and personalized ML*. 2020. URL: `https://github.com/epfl-iglobalhealth/coML-Personalized-v2`.

[6] Mary-Anne Hartley et al. "Predicting Ebola infection: A malaria-sensitive triage score for Ebola virus disease". In: *PLoS neglected tropical diseases* 11.2 (2017).

[7] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. "Secure Byzantine-Robust Machine Learning". In: *arXiv:2006.04747 [cs, stat]* (June 8, 2020). arXiv: `2006.04747`. URL: `http://arxiv.org/abs/2006.04747` (visited on 06/16/2020).

[8] Tom Hennigan et al. *Haiku: Sonnet for JAX*. Version 0.0.1. 2020. URL: `http://github.com/deepmind/dm-haiku`.

[9] Peter Kairouz et al. "Advances and Open Problems in Federated Learning". In: *arXiv:1912.04977 [cs, stat]* (Dec. 10, 2019). arXiv: `1912.04977`. URL: `http://arxiv.org/abs/1912.04977` (visited on 06/04/2020).

[10] Sai Praneeth Karimireddy. *JAX federated learning*. 2020. URL: `https://tinyurl.com/Karimireddy-Jax-FL`.

[11] Miaofeng Liu et al. *Reinforced Training Data Selection for Domain Adaptation*. 2019. DOI: `10.18653/v1/P19-1189`. URL: `https://www.aclweb.org/anthology/P19-1189` (visited on 03/27/2020).

[12] Miaofeng Liu et al. "Reinforced Training Data Selection for Domain Adaptation". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, 2019, pp. 1957–1968. DOI: `10.18653/v1/P19-1189`. URL: `https://www.aclweb.org/anthology/P19-1189` (visited on 03/27/2020).

[13] Yishay Mansour et al. "Three Approaches for Personalization with Applications to Federated Learning". In: *arXiv:2002.10619 [cs, stat]* (Feb. 24, 2020). arXiv: `2002.10619`. URL: `http://arxiv.org/abs/2002.10619` (visited on 06/03/2020).

[14] Ricardo Mendes and João P Vilela. "Privacy-preserving data mining: methods, metrics, and applications". In: *IEEE Access* 5 (2017), pp. 10562–10582.

[15] Mohamed Ndoye et al. "Collaborative privacy". In: (2020). URL: `https://www.mndoye.com/collaborativeprivacy.pdf`.

[16] OpenMined et al. *PySyft: A library for encrypted, privacy preserving machine learning*. 2020. URL: `https://github.com/OpenMined/PySyft`.

[17] Sebastian Raschka. *watermark: An IPython magic extension for printing date and time stamps, version numbers, and hardware information*. Version 2.0.2. 2019. URL: `https://pypi.org/project/watermark/`.

[18] Adam Richardson, Aris Filos-Ratsikas, and Boi Faltings. "Rewarding High-Quality Data via Influence Functions". In: *arXiv preprint arXiv:1908.11598* (2019).

[19] Theo Ryffel et al. "A generic framework for privacy preserving deep learning". In: *arXiv:1811.04017 [cs, stat]* (Nov. 13, 2018). arXiv: `1811.04017`. URL: `http://arxiv.org/abs/1811.04017` (visited on 06/13/2020).