April 9, 2023

Client Model Verification for Federated Learning

Evan Gronberg EGRONBE@JH.EDU
Engineering for Professionals
Johns Hopkins University
Baltimore, MD 21218

Abstract

Federated learning (FL) is a valuable architecture for allowing disparate parties to collaborate on machine learning models while keeping their respective datasets completely private and local. However, FL does have security flaws and is certainly not a zero-trust architecture. In particular, a plain FL architecture requires that we trust clients not to submit flawed models, whether those models be flawed intentionally or unintentionally. What is proposed here, then, is a method of verifying the performance of client models before aggregating them into the global model used by everyone. This method, called client model verification (CMV), is a statistical test for detecting outlier client models whose aberrant performance flags and excludes them from the global model.

1 Introduction

Federated learning (FL) is a valuable architecture for performing privacy-preserving machine learning with neural networks. The architecture has a one-to-many server-client relationship. The same neural network architecture is initialized on all parties. Each client trains a model using its local dataset. Each then pushes up its trained model to the server, along with the number of examples that were used to train the model. The server then performs a weighted aggregation of the client models. This process results in the global model, which leverages all of the data of all involved clients without revealing the clients' data to the server or to each other. This global model is then pushed out to all clients in the federation, concluding a round of training. The process then repeats.

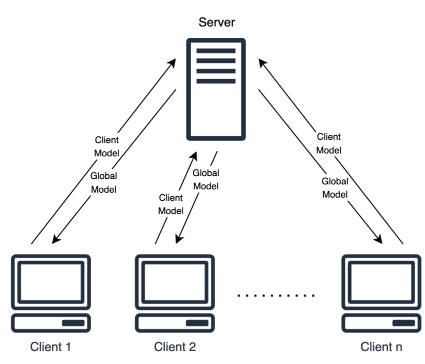

Figure 1: The basic architecture of federated learning.

The utility of this architecture is clear: disparate parties are able to collaborate on machine learning models in a way that respects data privacy. However, there are still security risks. A plain FL architecture assumes that both the server and the clients are trustworthy enough to not tamper with the federated learning process.

Existing research addressing this problem focuses on the trustworthiness of the server – that is, how we might minimize the amount of trust we have to place in it, especially in regards to privacy. See, for example, Huang et al. (2022) and Ren et al. (2022). However, existing research largely ignores the problem of assuming the trustworthiness of a federation's clients. There is some research addressing data drift amongst clients; see, for example, Duan et al. (2021) and Varno et al. (2022). However, there is a lack of research addressing malfunctioning or outright malicious clients. The goal of this paper, then, is to address this problem by introducing a method of verifying client models sent to the server.

2 Methods

For client model verification (CMV), a statistical method is proposed.

The server is given a test set to store locally. Before the server begins its weighted aggregation process, it evaluates each client model's accuracy on this test set. The accuracy of each model is then assigned a Z-score relative to the accuracy of all client models.

We then determine the 'weight' of each model using the number of training examples it proportionately contributed. We then take the absolute logarithm of these weights with the number of clients as its base. This logarithm, multiplied by the user-supplied average client Z-score threshold, is used as the tolerance for that client model's Z-score. That is, if the accuracy of a given client model differs by more than its tolerable Z-score, then the model is rejected.

Put simply, the more examples a model is trained on, the more strict its verification criteria.

Formally, CMV is defined as follows:

$$\forall c \in C, V(c) = \text{pass if round}(|\frac{a_c - \mu_A}{\sigma_A}|,\, 2) < z_c \text{ else fail}$$

where

- C is the set of all client models
- V is the verification test
- A is the set of accuracies for each client model
- Z is the set of the maximum allowable Z-scores for each client model

The set Z is calculated as follows:

$$Z = \{\forall c \in C, z_c = |\ \log_{|C|} \frac{n_c}{\sum n_i}\ | \cdot s\}$$

where

- N is the set of the numbers of training examples that were used to train each client
- s is the standard deviation threshold for the average client set by the user
  - Note: This is the one and only hyperparameter

2.1 Example

Let's say there are 10 client models trained on a total of 100,000 examples. Suppose that the first client model was trained on 10,000 examples. Furthermore, suppose the user set the average client standard deviation threshold to 1. Thus,

$$z_1 = |\ \log_{10} \frac{10,000}{100,000}\ | = |\ \log_{10} 0.1\ | = |\ -1\ | = 1$$

Now suppose that the accuracy of this model is 0.6, while the average accuracy every other client model is 0.8. Suppose also that the standard deviation of all the accuracies is 0.1. Thus,

$$V(1) = |\frac{0.6 - 0.8}{0.1}| = |\frac{-0.2}{0.1}| = |-2| = 2 \not< 1 \rightarrow \text{fail}$$

3 Results

All experiments below are run on the MNIST dataset using the example CNN provided in the Keras documentation: https://keras.io/examples/vision/mnist_convnet/

MNIST's 60,000 training examples are always divided equally among the clients, and each client trains for 5 epochs with a batch size of 128.

3.1 Effect on Performance of Bad Client Models with and without CMV

3.1.1 Experiment Design

To simulate malfunctioning/malicious client models, we will scramble a portion of the training labels in client datasets. We will evaluate the effect these malfunctioning/malicious clients have on performance over two variables:

1. The number of clients that are malfunctioning/malicious.
2. The number of training labels that have been scrambled on a given client.

We will hold the number of total clients constant at 10.

Once we have run these experiments without CMV, we will incorporate CMV and observe how well it preserves performance against malfunctioning/malicious clients. CMV's average client standard deviation threshold will be set to 1.0.

3.1.2 Experiment Results

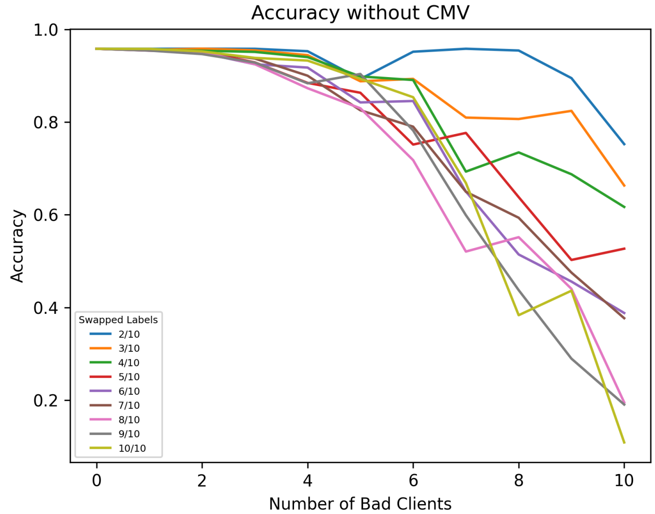

Figure 2: The accuracy of the global model as the number and low quality of bad clients increases **without** client model verification.

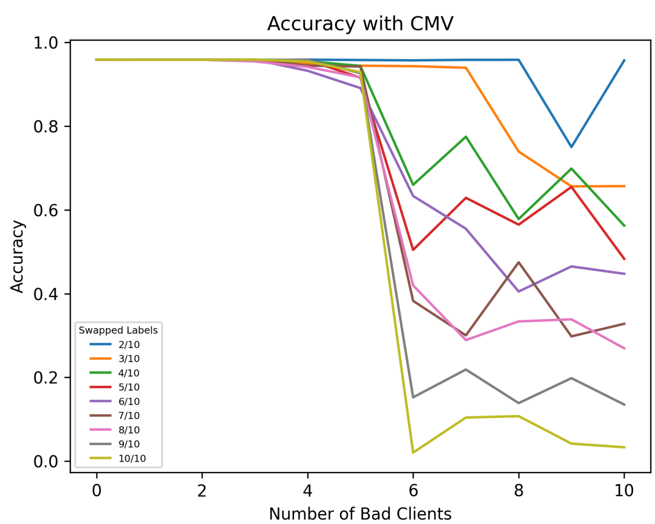

Figure 3: The accuracy of the global model as the number and low quality of bad clients increases **with** client model verification.

3.2 Minimum Number of Clients for Detecting Different Numbers of Bad Clients

3.2.1 Experiment Design

The more bad clients there are in a given round, the more the distribution of accuracy scores will be skewed toward bad clients. Thus, the more bad clients there are in a given round, the more good clients there must be to skew the distribution 'back' toward effectiveness.

We would thus like to determine how many total clients there must be in a federation to detect a given number of bad clients, where each bad client is half-scrambled. For this experiment, we will vary the average client standard deviation threshold, and there will be one line for each number of bad clients that we will test (1, 2, and 3). Each line will thus be plotted as the number of clients over the average client standard deviation threshold.

3.2.2 Experiment Results

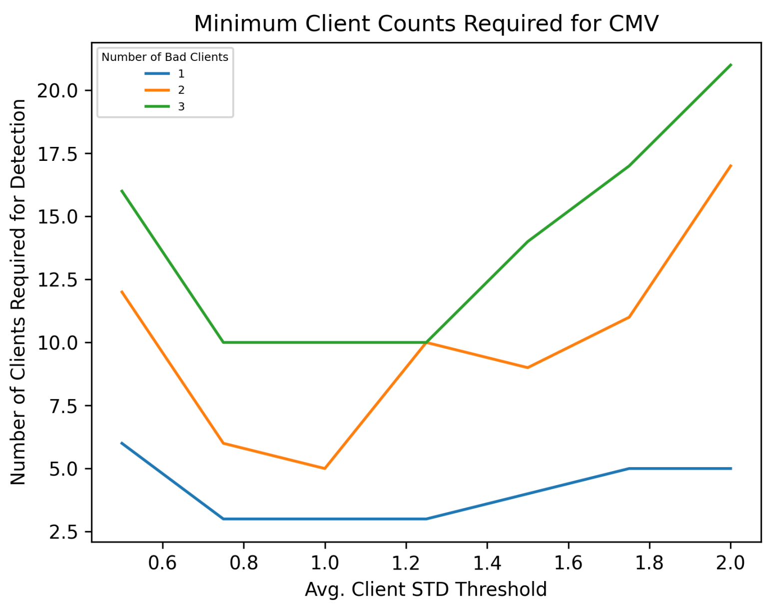

Figure 4: The minimum number of clients required to detect different numbers of bad clients at different average client standard deviation thresholds.

Note: A separate test was run to verify that 3 clients are required to detect 1 bad client regardless of how much scrambling has taken place on the bad client. This test is included in the `result_2.py` file of the project's GitHub repository; see Gronberg (2023).

4 Discussion

4.1 Result 1 Discussion

Result 1 demonstrates that, until the number of bad clients equals or exceeds the number of good clients, CMV is *very* effective at detecting and excluding the bad clients. In Figure 2, we see that performance begins to quickly degrade at 3 bad clients. In Figure 3, though, performance is maintained very well until the bad clients win equality or majority.

Once the number of bad clients equals or exceeds the number of good clients, CMV performance completely degrades, because it then excludes *good* clients. However, at this point, it is no longer reasonable to expect it to detect truly bad clients due to the 'majority vote' nature of the test.

4.2 Result 2 Discussion

Result 2 demonstrates an interesting trend. A lower average client standard deviation threshold actually requires *more* total clients to detect bad clients than when using the default of 1.0. This is not so that bad clients may be *excluded*, but so that good clients may be *included*. If there are not sufficient good clients to form a sufficiently 'tight' distribution, then most (or even all) clients, don't fall within standard deviations that are close enough to each other.

The minimum number of clients for effective detection comes at the default value of 1.0.

Once we increase from 1.0, then the number of clients required for bad client detection increases again. This is because, again, a sufficiently 'tight' distribution must be formed. This time, though, it must be sufficiently tight such that bad clients are quite distant from the center of the distribution – good clients will certainly be included, but there has to be sufficient consensus among them to flag bad clients as particularly 'far away.'

4.3 Limitations and Future Work

CMV is a simple method that certainly has limitations and room for improvement. Perhaps its greatest weakness is that it requires that the server has to have a test set, and this can be difficult to establish. We could have clients create and approve a test set collectively, but then clients have exact knowledge of the test set, and malicious clients may be able to submit faulty models that still perform well on that particular test set. The server could establish the test set on its own, but then the clients have to trust the server perhaps a little too much.

On that note, while we have greatly reduced the amount of trust that we have to place in the clients, we do still have to trust the server to apply the correct test set and perform the check and aggregation fairly. As an aside, trusted execution environments (TEEs) may be an effective tool for solving this problem, because they verify and ensure the integrity of the program running inside of them. See Drljaca (2021) for more information about TEEs.

Aside from addressing the above problems, future work on this topic might also include the following:

- Performing CMV on client model predictions themselves rather than on performance scores.
  - This has the benefit of exposing CMV to more data overall, and it also does not require that the server's test set be labeled.
- Verifying that clients are sufficiently unbiased for use cases that are known to manifest bias (e.g., loan approval).
- Collecting historic performance for CMV to compare clients against.

5 Conclusion

CMV proves to be a simple yet effective way to mitigate the impact of malfunctioning or malicious clients in a federated learning setting. Perhaps the most surprising result, though, is that, for a single round of training, whether or not we discard bad client models really only matters once there are multiple bad clients. The effect that one bad client has on performance is small. However, in the real world, a single malicious client would likely attempt to contribute bad models across multiple rounds of training, which would gradually reduce performance to a noticeably poor level. Fortunately, this is exactly what CMV prevents and what makes CMV quite useful.

References

Borko Drljaca. Trusted execution environments: Ensuring protection of data in use. `https://phoenixnap.com/blog/trusted-execution-environment`, 2021.

Moming Duan, Duo Liu, Xinyuan Ji, Yu Wu, Liang Liang, Xianzhang Chen, and Yujuan Tan. Flexible clustered federated learning for client-level data distribution shift. *CoRR*, abs/2108.09749, 2021. URL `https://arxiv.org/abs/2108.09749`.

Evan Gronberg. Client model verification for federated learning. `https://github.com/evangronberg/client-model-verification-for-fl`, 2023.

Chao Huang, Yanqing Yao, Xiaojun Zhang, Da Teng, Yingdong Wang, and Lei Zhou. Robust secure aggregation with lightweight verification for federated learning. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 582–589, 2022. doi: 10.1109/TrustCom56396.2022.00085.

Yanli Ren, Yerong Li, Guorui Feng, and Xinpeng Zhang. Privacy-enhanced and verification-traceable aggregation for federated learning. *IEEE Internet of Things Journal*, 9(24): 24933–24948, 2022. doi: 10.1109/JIOT.2022.3194930.

Farshid Varno, Marzie Saghayi, Laya Rafiee Sevyeri, Sharut Gupta, Stan Matwin, and Mohammad Havaei. Adabest: Minimizing client drift in federated learning via adaptive bias estimation, 2022.