# Detecting Adversarial Attacks on Federated Learning with Unbalanced Classes

CMPUT 644: Final Project Report. Fall 2022

Afia Afrin
aafrin@ualberta.ca

Javier Sales-Ortiz
salesort@ualberta.ca

Jihoon Og
og@ualberta.ca

## 1 INTRODUCTION

Federated learning (FL) enables multiple parties to jointly train a machine learning model without exchanging their local data; addressing critical issues such as data privacy, data security, data access rights, and access to heterogeneous data. While this added benefit has increased its popularity as a privacy-preserving machine learning technique, the distributed nature of FL gives rise to new threats caused by potentially malicious participants [8].

FL works by having clients train on a shared model using their own local dataset and send their learnt model parameters to a central server to be aggregated into a global model [9]. A malicious client could purposely send fictitious model parameters to the server causing a regression in the model accuracy, and/or an increase in the convergence time during training. An important component in improving FL systems is using effective detection algorithms in detecting these malicious clients so they can be removed from the training pool. Our main motivation for the work is to investigate the robustness of FL systems to model poisoning attacks leveraging the algorithms proposed in [2].

In this work, we aim to address the research gaps that exist in [2] and extend the work along a few additional dimensions which would allow us to investigate the resilience of the proposed poisoning attacks and detection systems in a variety of adversarial and practical settings. Our main contributions, in addition to reproducing the poisoning attacks and corresponding detection strategies proposed in [2], are as follows:

- We conducted experiments with two distinct distance metrics for one of the detection algorithms, and for the first time presented experimental results on the performance of detection metrics (section 3.3.2).
- We implemented adversaries with varying amounts of *poison*, and model the behavior of the proposed poisoning detection strategies.
- We proposed a smooth and incremental method to generate unbalanced datasets and evaluated the performance of *FedAvg* [10], and the proposed poisoning detection mechanisms on **unbalanced** datasets.

## 2 BACKGROUND AND RELATED WORK

Analyzing the adversarial robustness of FL environments has garnered considerable interest among scholars [11]. Adversaries against FL environments have been developed with the aim to affect model convergence [3], [6], poison the training datasets [1], and poison the local models in a targeted [2] or untargeted [15], [4] manner. In this project, we aim to extend one of the existing works in this area that focuses on introducing targeted poisoning in FL settings [2].

Novel and effective algorithms to produce model poisoning attacks against FL settings and to detect these attacks on the server side have been proposed in [2]. However, the experimentation has been done in a controlled environment, assuming each of the agents has a class-balanced training dataset from the same distribution, which is not always the case in practical settings. Moreover, no experimentation has been done to back up the proposed detection algorithms, which have only been explored conceptually. We are intrigued by the prospect of extending this work by investigating the effectiveness of the proposed *stealthy attack* strategy in terms of **model accuracy reduction** and **stealthiness** as well as analyzing the robustness of the proposed *detection methods* to a collection of **unbalanced** training datasets.

## 3 METHODOLOGY

In this section, we briefly explain all the necessary terminologies, concepts and algorithms that we have employed in our project.

### 3.1 FedAvg

We used Federated Averaging (*FedAvg*) for the model aggregation because the original authors [10] describe its usefulness in the non-IID case. Additionally, that was what was used in [2]. Federated Averaging works by having a fraction of clients train a shared model for a round of training. For each round each client $k$ who is selected for training will train the shared model using their local dataset for $l$ local

*Final Project Report, CMPUT 644, Fall 2022*

epochs. Equation (1) is used to aggregate the locally trained weights at the end of their last epoch into a single global model that is used for the next round of training.

$$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{1}$$

The weight for each client is based on their proportion of training data compared to the rest of all clients. However, all clients in our experiments have the same number of samples.

## 3.2 Model Poisoning Attacks

We implemented the following two model poisoning strategies that have been proposed in [2].

*3.2.1 Targeted model poisoning with explicit boosting:* In each round, the adversary optimizes a *malicious loss function,* $L_m$, which is designed to cause targeted misclassification. For this, the adversary uses an *auxiliary dataset*, which is a subset of the local dataset that the malicious agent possesses. The *auxiliary dataset* consists of data samples $\{x_i\}_{i=1}^R$ with true labels $\{y_i\}_{i=1}^R$ and adversarial labels $\{\tau_i\}_{i=1}^R$. Here, $R$ denotes the number of samples the adversary aims to misclassify. For the rest of the paper, we define $R$ as the *amount of poison* injected by an adversary.

At time step $t$, a malicious agent uses the global model parameters $W_G^t$ and computes its updated local model parameters, $W_m^{t+1}$ as:

$$W_m^{t+1} = W_G^t + \alpha \delta_m^{t+1} \tag{2}$$

where,

$$\delta_m^{t+1} = \underset{\delta}{\operatorname{argmin}} L_m \left( \{x_i, \tau_i\}_{i=1}^R; W_G^t + \delta \right) \tag{3}$$

Here, $L_m$ is the best-suited loss function for the classification or regression task. For our multi-class classification problem, we choose the *categorical cross-entropy* as $L_m$.

In order to negate the combined effect of all the benign agents, the malicious agent uses the idea of *explicit boosting* where they boost the malicious weight update, $\delta_m$ by some factor $\beta$ before adding it to the local model parameter, $W_m$. Following the methodology presented in [2], we choose $\beta = k$, where $k$ is the total number of agents.

Experimental results show a sudden drop in the global model's validation as well as test accuracy with the presence of an adversary crafting *targeted model poisoning* attack indicating that the proposed attack strategy is not stealthy enough and easily detectable. This outcome prompted the development of another stealthier adversary, named *alternating minimization attack*.

*3.2.2 Alternating minimization attack:* A stealthy version of the *targeted model poisoning* attack can be achieved by updating the malicious loss function, $L_m$. In order to obtain a stealthy $L_m$ two different notions of stealth, based on accuracy checking on the validation (or test) data, and weight update statistics have been introduced in [2]. The malicious objective function, $L_m$, is modified to account for these stealth metrics and the malicious objective function takes the form:

$$\begin{aligned} L_m^{stealthy} = L_m \left( \{x_i, \tau_i\}_{i=1}^R; W_G^t \right) + L \left( \{x_i, y_i\}_{i=1}^{n_m}; W_m^t \right) \\ + \rho \left( \| W_m^t - \hat{W}_{ben}^{t-1} \|_2 \right) \end{aligned} \tag{4}$$

Here, $\{x_i, y_i\}_{i=1}^{n_m}$ is the training data shard owned by the malicious agent and $\rho$ is a weight factor. In our experiments, we considered $\rho = 1$.

## 3.3 Detection Strategies

The detection of malicious clients is set as a binary classification problem. Membership to the positive class means that the client has been classified as malicious, whereas membership to the negative class is when a client is classified as benign. These are simple rule and heuristic-based classifiers with the only parameter being the classification threshold. The results are reported for a variety of threshold values as this parameter must be tuned manually.

The metrics used to measure the performance of the classification algorithms are as follows:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP} \\ \text{True positive rate } (TPR) = \text{ Recall} &= \frac{TP}{TP + FN} \\ \text{False positive rate } (FPR) &= \frac{FP}{FP + TN} \\ F_1 \text{ Score: } &= 2 \frac{Precision \cdot Recall}{Precision + Recall} \\ &= \frac{2TP}{2TP + FP + FN} \end{aligned}$$

*3.3.1 Detection based on validation accuracy.* This method examines how an agent's parameters affect the performance of the global model in order to identify whether or not that particular agent is malicious. For this strategy to work, a set of validation data needs to be stored on the central server. At the end of the $t^{th}$ FL round, the server compares the validation accuracy of $W_i^t$, the model obtained by using agent $i$, to the model obtained by aggregating all other parameters, $W_{G\setminus i}^t = \text{aggregate} \left( \{W_i^{t-1}\}_{i=1}^{k-1} \right)$. For any agent to be classified as *benign* by the server, it must satisfy the following

constraint:

$$\sum_{\{x_j,y_j\}\in D_{val}} \mathbf{1}\left[f\left(x_j; W_m^t\right) = y_j\right] - \mathbf{1}\left[f\left(x_j; w_{G\setminus m}^t\right) = y_j\right] < \gamma$$

The effectiveness of this detection strategy is heavily dependent on $\gamma$, which is a server-defined threshold to reject updates. An unnecessarily large threshold would fail to detect malicious clients but an overly small threshold would misclassify benign clients as malicious. Therefore, it is important to tune $\gamma$ to the appropriate value. However, the optimal range for $\gamma$ or how to tune it has not been presented in [2]. In this work, we aim to address this research gap by extending their work and experimenting with different threshold values.

*3.3.2 Detection based on distance ranges.* These methods, adapted from [2], aim to analyze the weights that each of the clients sends to the aggregation server in the hope that the weights from a *malicious* client exhibit different features than the ones from *benign* clients. These detection methods are, in essence, outlier detection methods where the malicious client is an outlier.

One key difference in our implementation is that we focus directly on weights ($W_m^t$) instead of weight updates ($\delta_m^t$). As described in Eqn. 2 the weight-updates can be recovered by: $\delta_m^{t+1} = \frac{W_m^{t+1} - W_G^t}{\alpha}$.

Particularly, the methods start by defining some measure of distance $d(\cdot,\cdot)$ between the weights of client $i$ and client $j$. In this project we experiment with two distance metrics:

- Euclidean distance: $d_E(w_i, w_j) = ||w_i - w_j||_2$ (5)
- Cosine dissimilarity: $d_c(w_i, w_j) = 1 - \frac{w_i \cdot w_j}{||w_i||_2 \cdot ||w_j||_2}$ (6)

Then, the methods calculate the pairwise distances of all the clients. Subsequently, a two ranges is calculated for each client $m$; with $m = 1, 2, 3, .., k$.

$$\begin{aligned} R_m &= [R_m^l, R_m^u] \\ &= \Big[\min_{i\in[k]\setminus m} d(w_m, w_i), \max_{i\in[k]\setminus m} d(w_m, w_i)\Big] \\ R_{[k]\setminus m} &= [R_{[k]\setminus m}^l, R_{[k]\setminus m}^u] \\ &= \Big[\min_{i,j\in[k]\setminus m;\, i\neq j} d(w_i, w_j), \max_{i,j\in[k]\setminus m;\, i\neq j} d(w_i, w_j)\Big] \end{aligned}$$

Therefore, $R_m$ is the range of distances between the client $m$ and all the other clients, while $R_{[k]\setminus m}$ is the range of pairwise distances among the rest of the clients, excluding $m$. Then, for any client $m$ to be classified as *benign* by the server, it must satisfy the following condition: [1]

$$\max(|R_m^l - R_{[k]\setminus m}^l|, |R_m^u - R_{[k]\setminus m}^u|) < \kappa \quad (7)$$

The choice of distance metric has an effect on the choice of the value $\kappa$ and the overall behavior of the classification. In this case, Euclidean distance is unbounded, while cosine dissimilarity ranges from 0 (exactly the same) to 1 (orthogonality or decorrelation) to 2 (exactly opposite). Both metrics can only output a positive number.

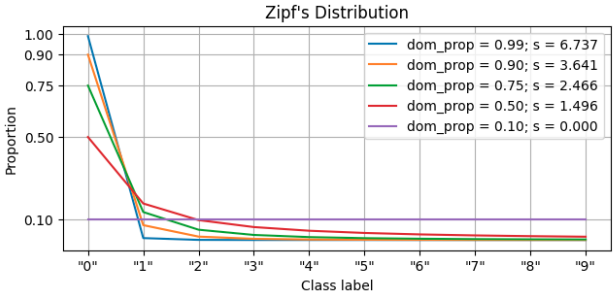



**Figure 1: Relative frequency of 10 classes ("0"-"9") within the first client at different dominant proportions**

## 3.4 Unbalanced Dataset

In order to expand the work done on [2], some of our experiments deal with unbalanced datasets. For this purpose, we define a methodical and incremental way to introduce class imbalance in each client's data shard.

For this purpose we use Zipf's distribution [12], which is a discrete power law distribution with a cutoff. Simply put, class $k$ out of $N$ will have a normalized frequency of $f$:

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^{N} 1/n^s} \quad (8)$$

One advantage of using this distribution is that it only depends on one parameter, $s$. However, this parameter is not interpretable. For this purpose, we define the imbalance metric of *dominant proportion* as the ratio between the most frequent class and the total number of samples.

$$\text{Dominant Proportion} = dom\_prop = \frac{\#\ majority\ class}{\#\ total\ samples} \quad (9)$$

Figure 1 shows the normalized class frequency within the first client. Note how the proportion for class label "0" corresponds to the *dominant proportion*. For the other clients, these frequencies are shifted accordingly.

[1] This equation has been modified from the original one, $\max(|R_m^l - R_{[k]\setminus m}^u|, |R_m^u - R_{[k]\setminus m}^l|) < \kappa$, presented in [2]. We have reached out to the original authors regarding the ineffectiveness of the original equation, but received no response. We are confident that the equation presented in this work does indeed trigger a positive when the two ranges being compared are distinct.

# 4 EXPERIMENTAL EVALUATION

This section describes our experimental setting and summarizes the results obtained from five different experiments[2].

## 4.1 Experimental setup

We used the framework Flower to simulate our FL environment. All 10 clients were used to train the global model using FedAvg. The model we used is LeNet-5 [7], the same one that has been proposed in [2]. Likewise, we used the same dataset Fashion-MNIST [14] for training and evaluation. The training set is comprised with 60,000 samples, with each client having 6,000 samples partitioned using the method described in subsection 3.4. The test set used to evaluate the performance of the aggregated model is comprised with 10,000 samples with equally distributed labels. For experiments involving malicious attacks, we ran the simulator for 20 rounds with 5 epochs per round. For the experiment comparing model performance with varying label imbalance, we ran the simulator for 40 rounds with 5 epochs per round.

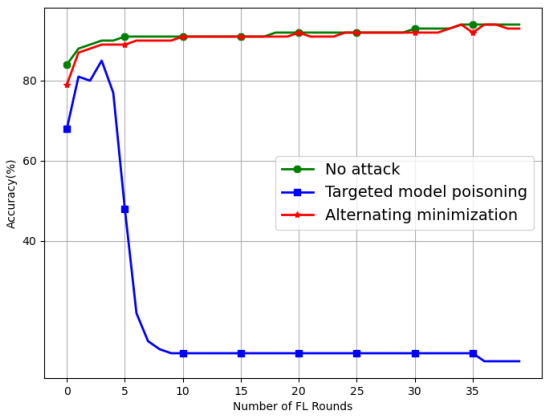



**Figure 2: Impact of poisoning attacks on global model's validation accuracy across different FL rounds**

## 4.2 Reproducing Malicious attacks

We reproduced the *targeted model poisoning* and the *alternating minimization attack* against the federated learning setting with 9 benign and 1 malicious agents and evaluated the shared model at the end of each training round. Fig 2 shows how the poisoning attacks affect the global model's accuracy over different number of FL rounds. As it can be seen, introducing *targeted model poisoning* drastically reduces the overall model accuracy, making it evident to the server that something suspicious is going on. The proposed *alternating minimization attack*, on the other hand, is more stealthy in its operation so that it has less of an impact on validation accuracy. This result follows the findings presented in [2] and guides us in designing our next experiments. Due to its' blatant nature, the *targeted model poisoning* strategy has been disregarded for our next experiments where we focus on analyzing the effectiveness of the proposed poisoning detection algorithms.

[2]The code for this project is open source and can be found online on https://github.com/afrin-afia/project_644

## 4.3 Effect of Malicious Attack on Model Accuracy

In this experiment, we implemented the *alternating minimization attack* with varying *amounts of poison*, $r \in \{0, 1, 10, 100, 1000, 3000\}$. Note that when $r = 3000$, it means that half of the *malicious client* samples are misclasified. The FL system employed 10 clients where each client undergoes 20 rounds of training with 5 epochs per round on balanced data ($dom\_prop = 0.1$).

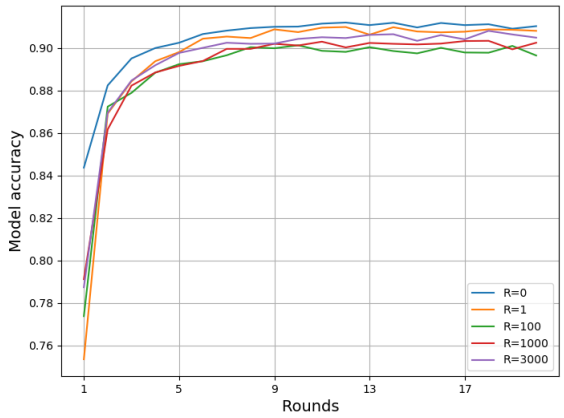



**Figure 3: Effect of *amount of poison* on global model's validation accuracy across different FL rounds**

Figure 3 shows how the malicious attack has only a small effect on the accuracy of the global model. This is because the *alternating minimization's* objective function takes the overall accuracy into account. This motivates the use of more rigorous detection methods compared to a simple visual inspection.

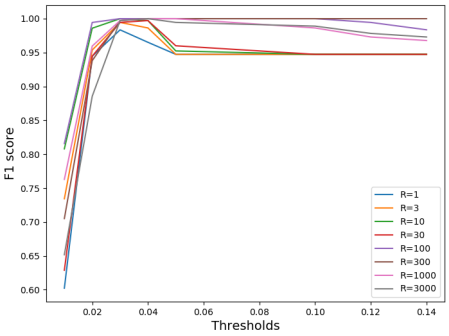
## 4.4 Detecting the stealthy malicious attack

In this experiment we trained an FL system incorporating an *alternating minimization attack* with the same varying *amounts of poison*, $R \in \{1, 3, 10, 30, 100, 300, 1000, 3000\}$.

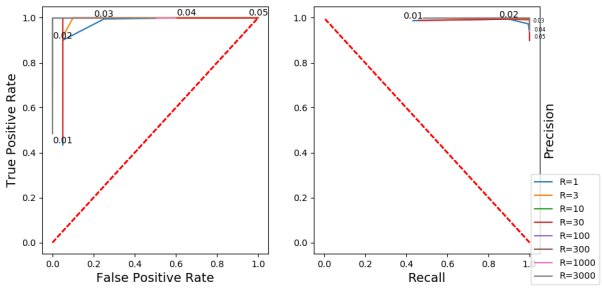
Similar to the previous experiment (in Section 4.3), the FL system employed 10 clients where each client undergoes 20 rounds of training with 5 epochs per round on balanced data ($dom\_prop = 0.1$). We saved the weights for each client

after each round for each value of $R$, and ran the detection methods described in Section 3.3 on those per-round weights. Meaning that for 20 rounds, the detection algorithms must correctly detect the *malicious* client 20 times, and the *benign* clients 180 times (9 *benign clients* $\times$ 20 *rounds*).

As the ratio of *malicious* clients to *benign* clients (1 : 9) is in itself unbalanced, we believe that it is misleading to report the naive accuracy (an agent that classifies all the clients as benign would be 90% accurate). Therefore we focus on $F_1$ score, Receiver Operating Characteristic (ROC) and Precision-Recall curves based on the metrics described in Subsection 3.3.




**(a) F1-score vs threshold values**




**(b) ROC curve and precision-recall curve**

**Figure 4: Threshold tuning: validation accuracy-based detection algorithm**

*4.4.1 Validation accuracy-based detection.* As described in section 3.3.1, the performance of the validation accuracy-based detection strategy is significantly dependent on the threshold parameter, $\gamma$. To determine the optimal $\gamma$, we employed the *alternating minimization attack* with varying *amount of poison, R* and evaluated the detection approach using a range of potential threshold values. Fig 4 illustrates our experimental findings on a **balanced dataset**. As depicted in fig 4(a) and 4(b), despite facing *heavily poisoned* attacks ($R > 1000$) the detection strategy's performance remains stable, which shows its robustness to the *amount of poison* induced by an adversary. However, the optimal $\gamma$ value ranges between 0.03 and 0.14, indicating that a server employing the *validation accuracy-based detection* approach should not tolerate more than 14% deviation in validation accuracy. Further, it should not reduce its tolerance level below 3% in order to maintain a decent *true positive to false positive* ratio.

*4.4.2 Distance ranges-based detection.* We applied the detection algorithm described in Equation 7 with Euclidean distance (Eqn. 5) and cosine dissimilarity (Eqn. 6). Here the threshold value is $\kappa$ from Equation 7. A smaller $\kappa$ means that the algorithm leans towards classifying more clients as *malicious*, and hence is susceptible to false positives. On the other hand, a larger $\kappa$ means that the algorithm is more lenient and is susceptible to false negatives.

For Euclidean distance we set $\kappa \in [0.2, 50]$; whereas for cosine dissimilarity $\kappa \in [0.02, 0.9]$

For the Euclidean distance F1-score, Fig. 5a, we can see that the range of effective $\kappa's$ decreases as the *amount of poison* increases. This effect might be due to a larger *amount of poison* introducing greater variability in the weights of all agents once all the agents are updated with the global weights after the first round. However, further experimentation is required to validate this hypothesis.

There still exists a range of approximately $\kappa \in [3, 10]$ where the detection algorithm performs well for every value of $R$ tested. The summary of the threshold ($\kappa$) values that produce an $F1-score > 0.9$ appears on Table 1. Then, Fig. 5c shows that the detection is imperfect only when $R = 3,000$.

A similar case happens for cosine dissimilarity, where the range of effective $\kappa's$ decreases as $R$ increases in 5b. As for Figure 5d, the algorithm can perfectly differentiate between benign and malicious clients.

## 4.5 Accuracy versus class imbalance

Figure 6 shows the aggregated model's performance in terms of accuracy for different values of dominance proportion for 40 rounds of training. The baseline is the line with the imbalance ratio of 0.1 or equal distribution of labels. With increasing values of imbalance we see a monotonic decrease in model accuracy for a given round of training. Figure 7 shows the distribution of classes for each client. The most significant of which is with a balance ratio of 0.99.

Interestingly, we only see a minor decrease in model performance for imbalance ratios 0.5 and 0.75. At 0.9 we see more of a jump compared to the previous two. A potential reason why a balance-ratio of 0.99 could lead to a significant regression could be caused by the model aggregation algorithm defined in section 3.1 not being sensitive to the dependencies of weights being influenced by different labels. That is, each client's model is overfitted to a particular class. When these are aggregated into the global model, a

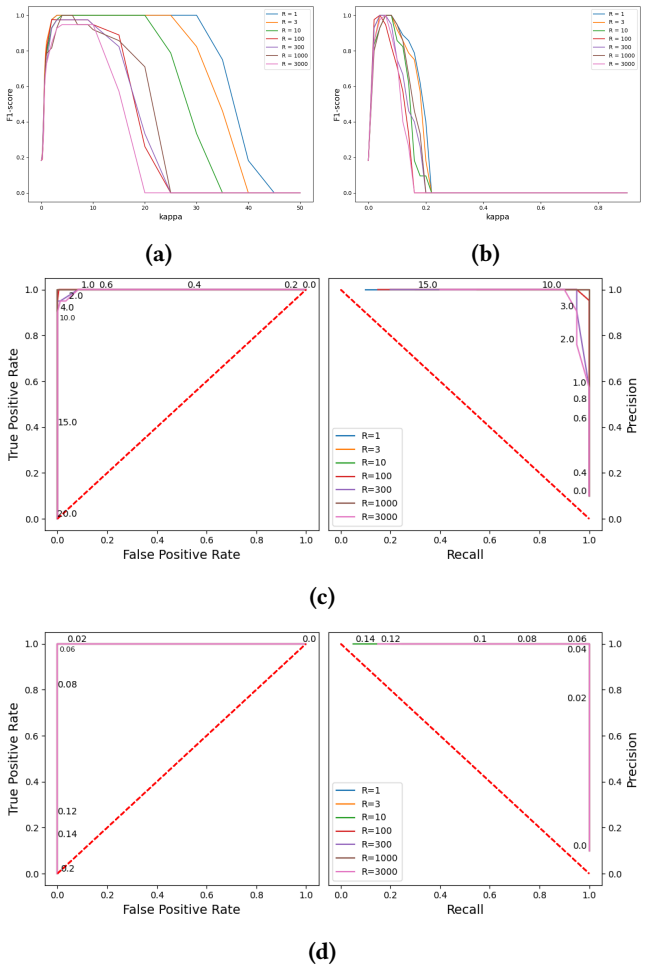



**Figure 5: Varying the *amount of poison*, performance of distance ranges-based detection algorithms.**
**(a) F1-score vs. $\kappa$ values, Euclidean distance.**
**(b) F1-score vs. $\kappa$ values, Cos dissimilarity.**
**(c) ROC and precision-recall curve, Euclidean distance.**
**(d) ROC and precision-recall curve, Cos dissimilarity.**
**On (c) and (d) numbers on the line indicate $\kappa$ values.**

simple *FedAvg* cannot extract useful weights from the clients' overfitted models.

## 4.6 Detecting stealthy attacks on unbalanced data

We generated sets of unbalanced data with various *dom_prop* ratio following the strategy presented in section 3.4 and investigated how the detection strategies behave with an *alternating minimization attack* with $R = 1$ (i.e. the malicious client only misclassifies one sample). The imbalance values used are $dom\_prop \in \{0.1, 0.5, 0.75, 0.9, 0.99\}$.

This section summarizes the corresponding findings.

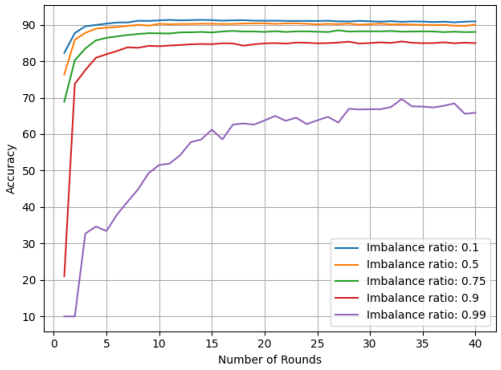



**Figure 6: Label Imbalance vs Accuracy**

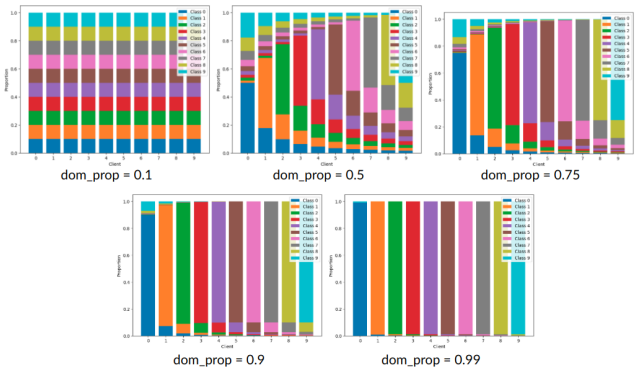
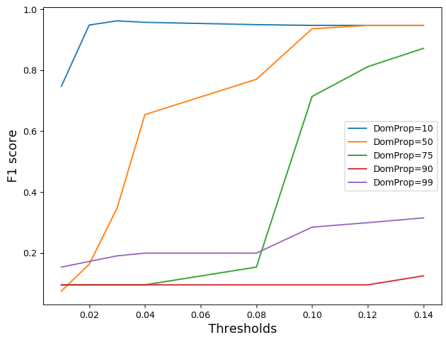



**Figure 7: Distribution of classes within each client at different values of *dom_prop***

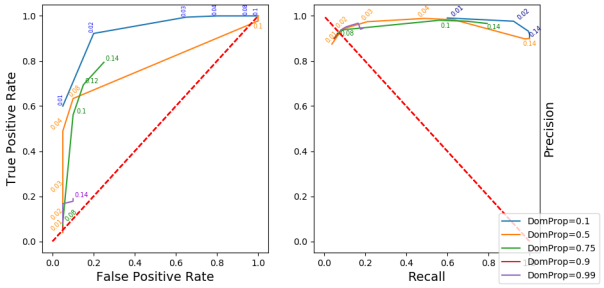
*4.6.1 Validation accuracy-based detection.* Figure 8 shows how *validation accuracy-based detection* strategy performs on class-imbalanced datasets. A close inspection of fig 8a reveals that the more unbalanced our training data pool is, the higher the threshold must be. Therefore, for this detection method to work on different settings and with different datasets, we need to manually calibrate the $\gamma$ value. For example, with *dom_prop* values above 0.50, optimal $\gamma$ needs to be significantly higher than 0.14, which is not in the optimal threshold range for an unbalanced dataset ($0.03 \leq \gamma \leq 0.14$). This observation indicates that there exists no optimal threshold value for this detection algorithm that offers a stable performance across datasets with varying imbalance ratios.

*4.6.2 Distance ranges-based detection.* As before, for Euclidean distance we set $\kappa \in [0.2, 50]$; whereas for cosine dissimilarity $\kappa \in [0.02, 0.9]$.

Some differences between the two distance metrics become apparent in this experiment. Note that for Euclidean distance the range for good $\kappa's$ decreases as *dom_prop* increases, as shown in Figure 9a. Conversely, this is not the case for cosine dissimilarity, where the range of good $\kappa's$ maintains its magnitude and shifts to the right as *dom_prop* increases as shown in 9b.




**(a) F1-score vs threshold values (unbalanced data, R=1)**




**(b) ROC curve and precision-recall curve (unbalanced data, R=1)**

**Figure 8: Threshold tuning: validation accuracy-based detection algorithm on unbalanced data**

As for the ROC and Precision-Recall curves (Figures 9c and 9d), given the appropriate $\kappa$ we can perfectly classify the *malicious* and the *benign* clients except when $dom\_prop = 0.99$. This parallels subsection 4.5, where *FedAvg* fails at the same $dom\_prop$. Excluding the extremely unbalanced case, these distance range-based methods are decidedly effective. Further, more sophisticated methods are needed to deal with aggregation and detection on an extremely unbalanced dataset.

## 4.7 Summary of threshold values

We present a table summarizing an appropriate choice of threshold values for the experiments in Subsection 4.2 and 4.6 for each detection method. Respectively, $\gamma$ is the threshold value for the validation accuracy method, and $\kappa$ for the distance ranges-based method. A good threshold is defined as being able to maintain $F_1$ score $\geq 0.9$ with all the $R$ or $dom\_prop$ values present in the respective experiment.

# 5 CONCLUSION

In this work, we analyzed the robustness of different FL settings to model poisoning adversaries. While our experimental findings regarding the *poisoning attack* strategies parallel the work presented in [2], we uncovered some intriguing results regarding the detection strategies that have

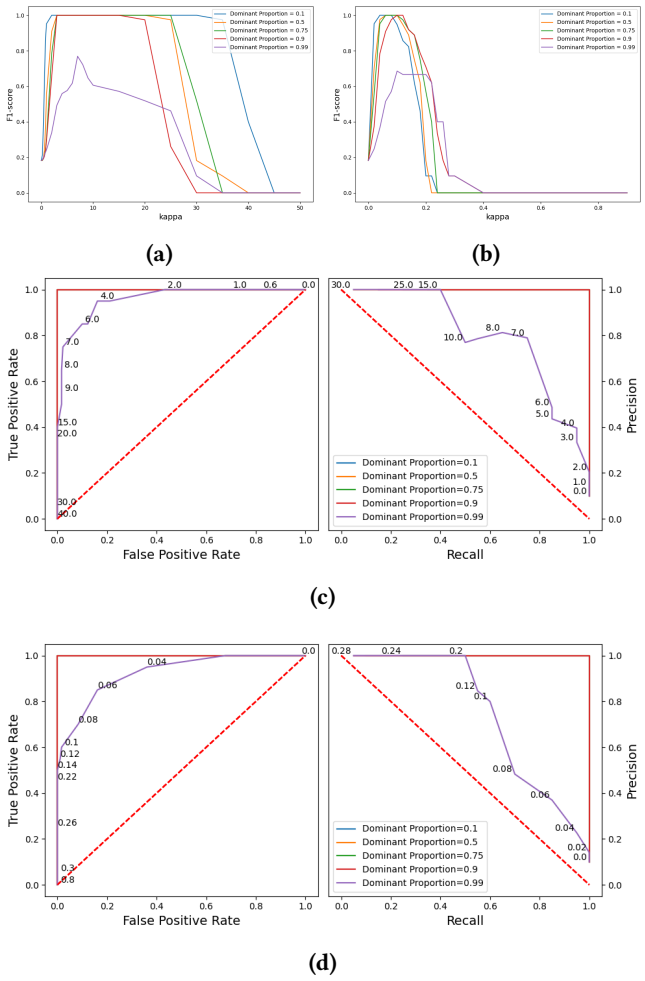



**Figure 9: Varying the dominant proportion, performance of distance ranges-based detection algorithms.**
**(a) F1-score vs. $\kappa$ values, Euclidean distance.**
**(b) F1-score vs. $\kappa$ values, Cos dissimilarity.**
**(c) ROC and precision-recall curve, Euclidean distance.**
**(d) ROC and precision-recall curve, Cos dissimilarity.**
**On (c) and (d) numbers on the line indicate $\kappa$ values.**

not been discussed in the original work. According to our findings, the weight statistics-based detection method seems promising against stealthy poisoning attacks. However, its effectiveness against other types of attacks requires further investigation. Moreover, validation accuracy does not appear to be a good poisoning indicator since attackers could easily train a malicious model to maintain the validation accuracy above a certain level and poison the model at the same time.

We tested FL with highly unbalanced datasets and found it extremely robust to data imbalance. However, FL, in every setting, is vulnerable to model poisoning attacks. Designing

| Experiment | Method | Min thr. | Max thr. |
|---|---|---|---|
| Balanced, varying R | Valid. Acc. | 0.03 | 0.14 |
| Balanced, varying R | Range, Eucl. | 3 | 10 |
| Balanced, varying R | Range, Cos | 0.04 | 0.06 |
| Unbalanced, R=1, Excluding all cases with domprop >0.50 | Valid. Acc. | 0.10 | 0.14 |
| Unbalanced, R=1. Exculding: dom_prop = 0.99 | Range, Eucl. | 3 | 20 |
| Unbalanced, R=1. Exculding: dom_prop = 0.99 | Range, Cos | 0.06 | 0.1 |

**Table 1: Minimum and maximum threshold (thr.) values that produce good results, an $F_1$ score $\geq 0.9$. The experiment in Subsection 4.4 is referred to as "Balanced, varying R", while the experiment in Subsection 4.6 is called "Unbalanced, R=1".**

robust FL systems that exhibit inherent protection against a range of poisoning attacks is a worthwhile research topic.

## 6 FUTURE WORK

This project can be extended to:

- Implement alternating minimization attacks with multiple malicious clients (adversaries), and in a combination with **data poisoning** strategies.
- Propose a detection algorithm that can self-tune it's threshold value, and different FL aggregating methods specializing in unbalanced datasets.
- Design a robust FL system, which incorporates effective detection schemes that can prevent or defend against a variety of attacks. The additional computational and memory cost of running the defense measures can also be investigated.
- Investigate the performance of other poisoning detection and defense algorithms such as PDGAN [17], FoolsGold [5], and AUROR [13].
- Test the robustness of the detection algorithms by the use of poisoned data generated by a generative adversarial network (GAN). Attacks on FL systems by the use of GANs to produce poisoned data have been done before [16]. Thus it is important to test the robustness of these algorithms against more sophisticated attacks.

## REFERENCES


[1] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2938–2948.

[2] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*. PMLR, 634–643.

[3] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Byzantine-tolerant machine learning. *arXiv preprint arXiv:1703.02757* (2017).

[4] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*. 1605–1622.

[5] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2018. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866* (2018).

[6] Rachid Guerraoui, Sébastien Rouault, et al. 2018. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*. PMLR, 3521–3530.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[8] Lingjuan Lyu, Han Yu, and Qiang Yang. 2020. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133* (2020).

[9] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[10] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR* abs/1602.05629 (2016). arXiv:1602.05629 http://arxiv.org/abs/1602.05629

[11] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. 2021. A survey on security and privacy of federated learning. *Future Generation Computer Systems* 115 (2021), 619–640.

[12] David M. W. Powers. 1998. Applications and Explanations of Zipf's Law. In *New Methods in Language Processing and Computational Natural Language Learning*. https://aclanthology.org/W98-1218

[13] Shiqi Shen, Shruti Tople, and Prateek Saxena. 2016. Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 508–519.

[14] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. https://doi.org/10.48550/ARXIV.1708.07747

[15] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2020. Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation. In *Uncertainty in Artificial Intelligence*. PMLR, 261–270.

[16] Jiale Zhang, Junjun Chen, Di Wu, Bing Chen, and Shui Yu. 2019. Poisoning Attack in Federated Learning using Generative Adversarial Nets. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 374–380. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00057

[17] Ying Zhao, Junjun Chen, Jiale Zhang, Di Wu, Jian Teng, and Shui Yu. 2019. Pdgan: A novel poisoning defense method in federated learning using generative adversarial network. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 595–609.

# A APPENDIX

## A.1 Team Responsibilities

All team members contributed equally to the creation of the project proposal, presentation, and final report. We divided the core tasks as fairly as possible and based on our assessment, all of us contributed equally to this project. The authors are listed in alphabetical order.

- **Afia Afrin:**
  - Outlined the research gaps in the existing literature and came up with the core project idea.
  - Implemented *targeted model poisoning* and *alternating minimization attack* and extended the baseline attack algorithms to include attacks with varying *poison level, R*.
  - Implemented the *validation accuracy-based detection* strategy.
  - Ran experiments section 4.4.1 and section 4.6.1 and reported the results.
- **Javier Sales-Ortiz:**
  - Came up with an unbalanced partitioning methodology using eq. (8), and a function to partition on *PyToch*.
  - Defined and implemented range-based detection algorithms, including correcting eq. (7) in [2].
  - Defined the distance metric of cosine dissimilarity, eq. (6).
  - Ran experiments section 4.4.2 and section 4.6.2, and reported the results.
- **Jihoon Og:**
  - Setup the Flower framework for all experiments including boilerplate code and custom model aggregations for saving the weights using FedAvg.
  - Optimized experiment settings for efficient remote executions on Snowball.
  - Download the dataset and model used in [2] and modified it to work with Flower.
  - Ran experiment described in section 4.5 which tests how class label imbalance affects accuracy and compiled the results.