Mitigating Data Poisoning Attacks On a Federated Learning Edge Computing Network

Ronald Doku and Danda B. Rawat
Data Science and Cybersecurity Center (DSC2)
Department of Electrical Engineering and Computer Science
Howard University, Washington, DC 20059, USA
rdoku@bison.howard.edu.com, danda.rawat@howard.edu

***Abstract*—Edge Computing (EC) has seen a continuous rise in its popularity as it provides a solution to the latency and communication issues associated with edge devices transferring data to remote servers. EC achieves this by bringing the cloud closer to edge devices. Even though EC does an excellent job of solving the latency and communication issues, it does not solve the privacy issues associated with users transferring personal data to the nearby edge server. Federated Learning (FL) is an approach that was introduced to solve the privacy issues associated with data transfers to distant servers. FL attempts to resolve this issue by bringing the code to the data, which goes against the traditional way of sending the data to remote servers. In FL, the data stays on the source device, and a Machine Learning (ML) model used to train the local data is brought to the end device instead. End devices train the ML model using local data and then send the model updates back to the server for aggregation. However, this process of asking random devices to train a model using its local data has potential risks such as a participant poisoning the model using malicious data for training to produce bogus parameters. In this paper, an approach to mitigate data poisoning attacks in a federated learning setting is investigated. The application of the approach is highlighted, and the practical and secure nature of this approach is illustrated as well using numerical results.**

I. Introduction

The goal of cloud computing is to deliver jobs to a remote network of powerful servers. These jobs usually involve the transfer of data for storage, management, or computation. A significant factor responsible for the intricacies associated with cloud computing is the heterogeneity of the data sources, which ranges from massive data centers to a wide variety of end-nodes. These data sources tend to be far away from the cloud, which presents a problem as the data obtained from these end-nodes will need to be uploaded and processed centrally in a remote cloud server. Examples of the data that could be collected from the end-devices are videos, measurements, photos, and location information which are aggregated at the remote server. Consequently, cloud computing faces issues such as network latency, efficiency, and the cost of transferring such extensive data to a remote server. These issues have necessitated the rise of edge computing and its variants as a vital tool in the quest to provide better services to clients. Edge computing [1]–[4] entails pushing the data for analysis closer to the source as it is a less expensive undertaking, especially when moving this across a wide area network to a remote server.

Furthermore, uploading the data over a long distance can expose the data to security issues. Edge computing endeavors to eliminate some of the risks of these attacks as it involves shortening the distance the data travels. However, this still does not present a foolproof solution to data security. A more dependable solution is to get the data to stay on the device for processing. FL has been proposed by Google researchers in an attempt to ensure data privacy in a distributed machine learning over multiple devices setting. In FL, the data does not have to leave the data source. The code comes to the data instead, which makes it a reliable way of optimizing privacy issues.

However, data poisoning in FL may arise as a result of participants training their data locally and then sending the model updates to the edge server for aggregation. In such a scenario, it is challenging for an edge-server to examine the data used for model training thoroughly. Consequently, an adversary can undermine the global model on the edge-server by introducing aggregates trained on dirty-labeled data. The work in [5] pointed out that dirty-label data poisoning attacks tend to produce high misclassifications in deep learning processes, with up to 90%, when an adversary introduces relatively few dirty-label samples (around 50) to the training dataset. As a result, work in [6] pointed out data poisoning attacks in FL as an issue that needs immediate addressing.

Label-flipping [7] and backdoor attacks [8], [5] are examples of data poisoning attacks that occur in a centralized ML setting. Robust losses [9] and anomaly detection [10] are approaches employed to mitigate these poisoning attacks in centralized scenarios. However, these approaches are not feasible in an FL setting as they require exclusive access to the training data. A constraint in FL is that the server or ML practitioners must not have any contact with the client's data, except through an ML model. The edge server should only receive model parameters. This way, FL allows multiple entities to train a model without needing to share sensitive data directly. Consequently, the centralized approach of having access to a dataset in order to mitigate poisoning attacks would

This work is partly supported by the U.S. NSF under grants CNS/SaTC 2039583 and NSF-1828811, and by the DoD Center of Excellence in AI and Machine Learning (CoE-AIML) at Howard University under Contract Number W911NF-20-2-0277 with the U.S. Army Research Laboratory.

defeat the purpose of FL if such measures were employed.

In this approach, a solution is devised that attempts to mitigate data poisoning attacks in an FL-EC network. We focus solely on text data for this research. An advantage a decentralized network has over its centralized counterpart is that the origins of the data are known, which provides an opportunity for the data to be vetted before it is used for training. Thus, our approach requires the vetting of a client's data prior to it being trained by the model. However, the goal of FL is to ensure data privacy by making sure the edge server does not come into contact with the data directly. The challenge here is to vet the data without compromising the security of the client's data and the values on which FL was established. To this end, we employ a mediator (aptly termed the facilitator), that serves as a liaison between the client node and the edge server. The facilitator is responsible for data vetting. The novelty of our approach lies in the manner in which the data is vetted and the introduction of a facilitator. We provide numerical evidence based on our evaluations and tests to prove that this approach, indeed, works to ensure higher security and improves the prediction rate of the model.

A. Background and Related Work

Currently, most applications and services rely on machine learning technologies to derive actionable insights from data. However, ML systems are vulnerable to adversarial ML attacks that subsequently compromise predictions. EC is a thriving sector that relies on the data sent from edge devices to train ML models for prediction services. However, data poisoning attacks are viewed as an emerging security threat that compromise such services [11]. An example of a data poisoning attack is optimal poisoning attacks, where an adversary injects malicious points in the training set which consequently increases the test classification error, with the adversary learning the parameters of the model. This is achieved by the minimization of a loss function processed on compromised data. This approach has been used against classic binary ML algorithms such as Support Vector Machines (SVM) [7], logistic regression [12], and embedded feature selection [13]. The algorithm proposed in [14] measures the effect each instance of the training set has on the performance of the learning algorithms, which works more adequately with smaller datasets in comparison to larger ones. Another method employs an outlier detection scheme that recognizes and eliminates anomaly samples [15]. However, it is limited in defending against label flipping attacks. In [16], they address the consequences of data poisoning attacks on an FL system beset with Sybils. In these attacks, adversaries produce multiple malicious participants to carry out attacks by utilizing poisoned data for model training. They employ a novel defense strategy (FoolsGold) where they distinguish legitimate participants from malicious ones based on their updated gradients. In this work, the focus is primarily on label-flipping attacks, and the proactive rather than a reactive measure taken to mitigate it. Since the FL server is prohibited from having access to the training data, we employ a liaison that handles the data vetting process.

II. Proposed Approach

Machine Learning in all of its glory can be reduced to a basic form which is usually a simple equation such as:

$$Y = sgn\left(w^T x + b\right) \tag{1}$$

And whatever prediction is made depends on the result of Equation 1 which could either be smaller than 0 (negative outcome) or greater than 0 (positive outcome). The result of the prediction is dependent on the representation for x.

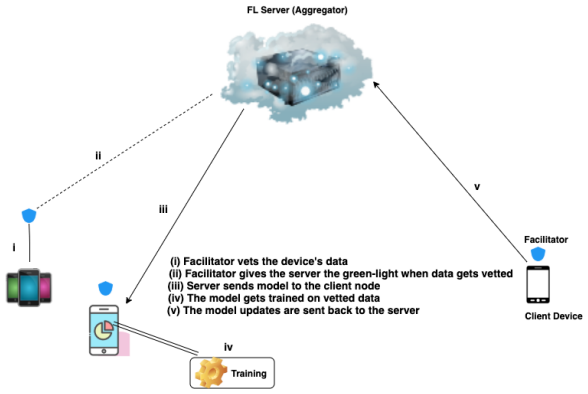

Figure 1. Data Vetting in the FL Process.

In this work, this simple but yet powerful idea is taken and employed in the data vetting process. To vet data (determine whether a potential dataset is poisoned or not), Equation 1 (SVM's equation) is employed. We utilize SVM for vetting data because an FL setting provides an advantage over it's centralized counterpart in the sense that, it allows for a proactive approach to data poisoning mitigation. This is because most of the work done in centralized ML systems focus on gathering data from the wild (usually unknown sources). However, the central server would still have access to the dataset. In FL, it is an ML model that has access to the dataset, not the edge server. However, the data sources are known which in turn provides the possibility of vetting the data before it is trained on an ML model. It is imperative to ensure the vetting process does not compromise the client's sensitive data (the foundation on which FL is built on). To this end, a facilitator is employed to serve as the link between the client and the edge server. The facilitator is a lightweight program that possesses an SVM model and other essential features it requires to effectively vet an end-device's dataset in a fair and privacy secure manner. Fig. 1 provides a visual representation of the process.

III. System Model

Our approach uses an SVM model to determine whether the dataset belonging to an end-device has been dirty-labeled or not. SVM [17] is a large margin classifier that aims to find an optimal hyperplane during a binary classification process that divides two classes in an n-dimensional space. A larger margin ensures the SVM classifier will have a better chance of generalizing future datasets. The goal of SVM is to find a vector w such that the dot product with all positives is above a $+1$ and the dot product with all the negatives is under -1.

This turns to be a minimization problem as the goal is to minimize the weight vector, which is the same as maximizing the margin. This is represented in the following equations below.

$$\sum_i w_i d_{i,j} \geq +1 \tag{2}$$

$$\sum_i w_i d_{i,j} \leq -1 \tag{3}$$

$$min\|\vec{w}\| \tag{4}$$

If we find the w, w has a particular form which is:

$$\vec{w} = \sum_{j\varepsilon+} a_j \vec{d}_j - \sum_{j\varepsilon-} a_j \vec{d}_j \tag{5}$$

Equation 5 consists of two parts. The first part is a linear combination of documents in the positive class minus a linear combination of documents in the negative class with some weights. The goal of the SVM classifier is to find a w that satisfies the conditions above. For a misclassification to happen in an SVM classifier, the datapoint must land on the wrong side of the separating hyperplane. This is called a classification error. However, if a data point is found inside the margin, it is termed a margin error. The total error *(T)* of the model is calculated as the sum of the classification error plus the margin error. A soft margin SVM model is employed to allow some classification and margin errors to occur. The number of errors allowed is controlled by the C parameter. The objective function the soft margin SVM tries to minimize is:

$$\frac{1}{2}\|w\|^2 + C\sum_i \max(0, 1 - y_i(w^\intercal x_i + b)) \tag{6}$$

A small C parameter is used to allow for more errors. T plays a pivotal role in the determination of whether a dataset is poisoned or not poisoned. For the vetting process, we calculate T for both poisoned and non-poisoned datasets. The poisoned and non-poisoned datasets are averaged to get a value we call the midpoint. Equation 7 illustrates how the midpoint is calculated. *TP* represents the total averaged error for the poisoned datasets, and *TN* is the total average error for the non-poisoned datasets. The midpoint serves as the boundary, and a future dataset that falls within the *TP* range during the vetting process is classified as a poisoned dataset. The idea here is to find the average total error of various poisoned datasets during training, and then use that to classify future datasets. Thus this approach requires the SVM model to be trained thoroughly on poisoned and non-poisoned datasets before it is deployed to a client node via the facilitator.

$$Midpoint = \frac{\sum \frac{TP_i}{n} + \sum \frac{TP_j}{k}}{2} \tag{7}$$

After the SVM model is successfully trained on the dataset, instances of the SVM model are assigned to a facilitator. A facilitator is deployed to every new node that joins the FL network. The main role of the SVM model the facilitator holds is to generate the total error produced by a dataset to classify whether the dataset is poisoned or not.

A. Feature Engineering

The representation of data is primarily the most important factor in machine learning. As such to effectively predict poisoned data, our goal is to train an SVM model on poisoned and non-poisoned data. To achieve this, we need to engineer an effective, consistent, and reliable feature set from the dataset on which the model would be trained and tested. We chose to use the IMDB dataset for this research because [18] discovered that this dataset was more susceptible to adversarial attacks than an image dataset when the bag-of-words features are utilized. Thus, we decided to engineer multiple feature sets that are not solely dependent on the bag-of-words approach. To this end, the main part of the vetting process focuses on how we engineer these feature sets. The attribute-value pairs that would make up the feature sets are a proof of common interest score, the device's reputation in the network and the topics of the dataset. We train the SVM using these d-dimensional attributes. The whole process begins with the SVM model being trained on a dataset with poisoned and non-poisoned labels. This dataset is supposed to have the same qualities of the prototypical data the FL model will be trained on in the real world.

1) Facilitator: Every node in the network is assigned a facilitator. The facilitator serves as the liaison between the edge server and the end device. The facilitator's job is to ensure that the data the global model will be trained on is not poisoned. The facilitator possesses the SVM model that will detect the total error from the client's data. The facilitator also extracts the needed features from the client's dataset for the SVM model. To ensure security and privacy, whatever features the facilitator generates is stored on the client device. The client device provides a link to the facilitator that allows it to access the generated features and then utilize it to determine if the dataset is vetted or not. If the client opts to discontinue the process, it just deletes the link to the feature set the facilitator had access to. After unlinking, the client can choose to overwrite this data with garbage as well for extra security [19], [20]. This ensures the facilitator now has no access to the features derived from the client data after it is removed. Whatever features the facilitator generates, it is stored on the client device. The client data provides a link to the facilitator that allows it to access the generated features and then utilize it to determine if the dataset is vetted or not. If the client opts to discontinue the process, it deletes the link to feature set the facilitator had access to. After unlinking, the client can choose to overwrite this data with garbage as well for extra security [19], [20]. This ensures the facilitator now has no access to the features derived from the client data after it is removed. The only communication the facilitator has with edge server is a 'yes' or 'no' response to whether the data is poisoned or not.

2) Proof of Common Interest: The Proof of Common Interest (PoCI) [21] is a concept that addresses the quantity over quality issue in data science. It attempts to solve this by devising a an consensus mechanism known as the PoCI to

vet data. In this work, we implement the PoCI in [21], and use it as an attribute in our feature engineering process. The PoCI as an attribute ensures that the data of non-malicious end devices will usually tend to share similar interests, and we take advantage of that fact.

The PoCI is calculated by using a unique hash function known as the MinHash of the dataset. The MinHash is a unique signature of a node's document that can be used to determine the similarities between two documents. To compute the PoCI, we need to find the similarities between the MinHash of a data owner's data and the minhashes of the datasets that were used for training (dirty-labeled and correctly labeled). The edge server sends random minhashes of selected dirty-labeled and rightfully labeled data to the facilitator. The data owner calculates it's own minhash. After the minhash calculation, we calculate the minhash by counting the number of components present in the two signatures. That gives the similarity score for the comparison of any two documents. We calculate the minhashes of the dirty labeled and correctly labeled data.

The PoCI process demands the calculation of a unique hash function known as the MinHash of the data owner's data. To compute the MinHash, we utilize data mining methods such as shingling and Jaccard Similarity. The shingling of documents involves treating documents as a set of short strings. This way, the documents that share common sub-strings are perceived as similar. Shingling solves this by transforming a document into multiple substrings of length k that is present within the document. As such, documents are represented as a set of k-shingles. The length k needs to be picked according to the size of the document. We pick a shingle size of $k = 4$. After picking the shingle size, we introduce MinHashing. Like other hashing techniques, MinHashing works by converting a document of any size into a specific size. However, MinHashing can specifically return a fixed-size numeric signature for documents. We can use this numeric signature to calculate the similarities between the two documents. This is done through the help of Jaccard Similarity. We can find the similarity between two documents A and B by performing the Jaccard Similarity by discovering the relative size of their intersection. When documents are presented as a set of shingles, we can use the Jaccard Index to measure the similarity. The Jaccard Similarity can be applied to MinHashes as well. The Jaccard Similarity between the MinHashes of two documents A (MH_A) and B (MH_B) and is defined as:

$$J(MH_A, MH_B) = \frac{|MH_A \cap MH_B|}{|MH_A \cup MH_B|} \Bigg| \tag{8}$$

MinHashing uses randomized algorithms to estimate the Jaccard Similarity between documents. The steps below show the MinHashing process:

Step 1:

1) Break down the ledger into a set of shingles.
2) Calculate the hash value for every shingle.
3) Store the minimum hash value found in step 2.
4) Repeat steps 2 and 3 with different hash algorithms 199 more times to get a total of 200 min hash values (MinHash signature).

To compute the PoCI, we need to find the similarities between the MinHash of the data owners' and the minhashes the facilitator currently possesses. This is achieved by counting the number of signature components in which they match. That gives the similarity score for the comparison of any two documents. The formula for calculating the MinHash is expressed as:

$$h_\pi(C) = \min_\pi \pi(C) \tag{9}$$

where C represents a document. After we get the POCI score of the data, that becomes the first feature we need.

3) Topic Models: The next feature we extract is the topic model. We use the Latent Dirichlet Algorithm (LDA) to achieve this. In an LDA model, the observable features the model sees are the words that appear in the documents. Other parameters are hidden/latent (inferred). One of those parameters is a topic that is assigned to each word, thus making every document a mixture of such topics. The goal of the model is to figure out how such document collection could have been generated in the first place. LDA produces a file that contains all the topics consisting of words with the probabilities of them belonging to that topic. The output from the LDA model is a k topic document with each topic represented by a group of words. We run the LDA model on both training data sets, and then come up with the topics present in each dataset. We then run the same LDA model on each dataset to come up with the probability that it belongs to dirty-labeled and correctly labeled given the topics it generated.

LDA works by assuming a document constitutes multiple topics i.e., $P(z|d)$. Each topic is a distribution over terms in a vocabulary i.e., $P(t|z)$. We operate under the assumption that there is a fixed vocabulary. LDA suggests documents are probability distributions over latent topics and topics are probability distributions over words. This means that according to LDA, every document contains a number of topics. Each topic has a distribution of words associated with it. The important aspect here is that LDA works with probability distributions rather than with strict word frequencies. Thus while other bag-of-words models may focus on the most frequently occurring words in a document, LDA takes a more holistic approach by concentrating on the distribution of words across topics. LDA can be formally written as :

$$P(t_i|d) = \sum_{j=1}^{z} P(t_i|z_i = j)P(z_i = j|d) \tag{10}$$

where $P(t_i|d)$ can be expressed as the probability of the i^{th} term for a given document d and z_i is the latent topic. $P(t_i|z_i = j)$ is also expressed as the probability of t_i within topic j. Furthermore, $P(z_i = j|d)$ is the likelihood of selecting a term from topic j in the document. The number of latent topics Z has to be defined in advance and allows to adapt the degree of specialization of the latent topics. LDA approximates the topic–term distribution $P(t|z)$ and the document–topic

distribution $P(z|d)$ from an unlabeled corpus of documents using Dirichlet priors for the distributions and a fixed number of topics. Gibbs sampling is one viable method to achieve this: It iterates multiple times over each term t_i in document d_i, and samples a new topic j for the term based on the probability $P(z_i = j|t_i, d_i, z_{-i})$ based on Equation 2, until the LDA model parameters converge.

$$P(z_i = j|t_i, d_i, z_{-i}) \propto \frac{C^{TZ}_{t_{ij}} + \beta}{\sum_t C^{TZ}_{t_{ij}} + T\beta} \frac{C^{DZ}_{d_{ij}} + \alpha}{\sum_z C^{DZ}_{d_{iz}} + Z\alpha} \tag{11}$$

C^{TZ} keeps a tally of all topic–term assignments, C^{DZ} counts the document–topic assignments, z_{-i} represents all topic–term and document–topic assignments except the current assignment z_i for term t_i, and α and β are the hyperparameters for the Dirichlet priors, working as smoothing parameters for the counts. Based on the counts the posterior probabilities in Equation 1 can be estimated as follows:

$$P(t_i|z_i = j) = \frac{C^{TZ}_{t_{ij}} + \beta}{\sum_t C^{TZ}_{t_{ij}} + T\beta} \tag{12}$$

$$P(t_i|z_i = j|d_i) = \frac{C^{DZ}_{d_{ij}} + \alpha}{\sum_z C^{DZ}_{d_{iz}} + Z\alpha} \tag{13}$$

The topics generated can then become a simple document that the facilitator can use as a training set for a Naive Bayes classifier (NB). For this feature to be extracted in a client device, we get the topics from the client dataset, and then generate the probability of the dataset belonging to either the poisoned or non-poisoned dataset. When we set the parameter k to 10, the LDA model in the facilitator derives 10 topics from the dataset. The next step is to determine the probability of these generated topics belonging to which class (poisoned or non-poisoned) (NB).The facilitator will possess a lightweight training (100 topics for each class) which it can use to predict the probabilities of the dataset belonging to a poisoned or non-poisoned class. As a general statement, we can state Baye's theorem as follows:

$$P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})} \tag{14}$$

The data are represented by $\mathbf{D}$ and parameters are represented by θ.

4) Reputation: Another feature we extract is the reputation of the node in the system, we take into account the number of times it passed the vetting process during it's time in the network. At the initial step, each node has a reputation score of 0.5. The reputation score is calculated by choosing a *k-bit* length vector. In our case, we chose k to be 16. Hence, we possess a 16-bit length binary vector where a bit of 1 in the 16-bit sequence denotes a node's dataset has successfully been vetted, and 0 denoting failing the vetting stage.

Attached to each reputation vector is a number m, which represents the number of most significant bits. The m significant bits are found by counting bits to the right. To evaluate the score of a reputation vector, we count up to the m^{th} significant bit and then convert it to an integer. After this is done, we divide it by 2^m. For instance, if we want to get the score for a reputation vector for a node i from node j, with the reputation vector of rv_{ij} = 1110011000000000 where m= 7. The score it represents can be calculated as: $\frac{(1110011)}{2^7} = \frac{115}{128} = 0.8984$. This gives us a value between [0 to 1). Everyone in the network is expected to calculate this value. We also keep track of a non-reputation vector, which is just the complement of the reputation-vector. The facilitator communicates with other facilitators in the network for the propagation of this information. Initially, when a node has just joined the network, 0.5 is the default score for that node. The score gets updated after each iteration. A node gets kicked out of the network if after 16 iterations (vetting processes0, it's reputation score is lower than a set threshold. The whole process begins again for legitimate nodes after 16 data vetting processes.

IV. Testing and Evaluation

After the feature extraction stage, the next step is to train the model using the features generated. For the initial model training, we assigned random values for reputations based on what good and bad reputation scores were determined to be. We carefully curated a prototypical poisoned and non-poisoned dataset. This was important because we needed a representation of what a good and bad data was in order to accurately generate the POCI and topic modeling features. We perform experiments on the IMDB review sentiment dataset. In the experiment, we performed uniform random flips on the data which results in the case where instances of the training data are uniformly chosen at random and their labels are flipped. We chose to use the IMDB dataset because [18] discovered that this dataset was more susceptible to adversarial attacks than an image dataset when the bag-of-words features are utilized. Hence, we decided to engineer the features to measure the effectiveness of our approach. The IMDB review sentiment dataset has 25,000 training examples and 25,000 test instances, which has been equally divided between "positive" and "negative". To effectively pick the number of instances to flip the labels on, we computed how many instances in the data we have to label-flip in order to diminish the model's prediction. This number turned out to be 18% of the sample size. Thus, 150 for a dataset that had 800 instances. The goal of this experiment during the initial phase was to determine the midpoint. To accomplish this, we split the dataset into 25 batches. Each batch had 2000 instances of the dataset. For each batch, the dataset was split into two groups. The first group had the poisoned dataset and whilst the second group had the non-poisoned dataset. Each group had 1000 instances each. We split the dataset into training and testing. The training dataset had 700 training instances, and a 300 instance test set. For the poisoned group, we flipped the label for at least 132 (18%) instances. For the test data, we flip the label poison for at least 52 instances out of the 300. We then run the SVM classifier on it. The goal was to determine T. We then proceed to the non-poisoned group where we set aside 700 training

data and 300 test data as well. We run the SVM model and calculate the error. Thus for the first batch, we have TP for the poisoned batch and TN for the non-poisoned batch. We repeated the process for the other 24 batches and recorded their total error for each group. After the experiments, we ended up with 25 TPs and TNs. We derive the midpoint from these two results. We run experiments to measure how accurately we can determine poisoned or non-poisoned data using this process. We generated 25 randomized datasets (poisoned and non-poisoned). For each dataset, the SVM model determines its total error. If the total error derived is less than the midpoint, we predict that the dataset is poisoned. The accuracy score was 0.72. Table I shows the precision and recall. We also plot the misdetection rate based on the percentage of falsified data in the dataset (Fig. 2).

Table I
PRECISION AND RECALL

	Precision	Recall	F1-score
0	0.77	0.71	0.74
1	0.67	0.73	0.70

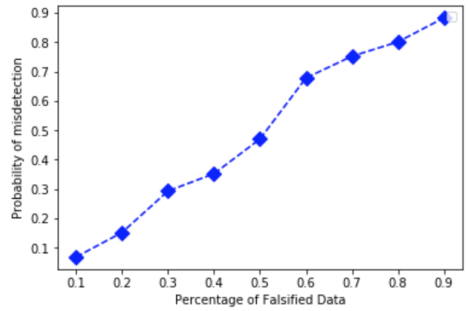

Figure 2. Misdetection variation for % of falsified data.

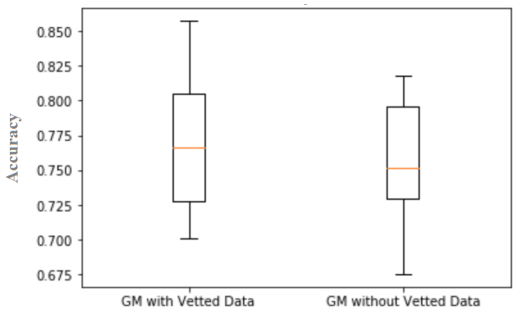

Figure 3. Comparing Model Performance

Our approach performed poorly when the percentage of the flipped label is high, and did reasonably well when the percentage dropped. To test how well our method performed, we generated various data samples each with varying degrees of poisoned data. We had a test group where we train the data on a model after it has passed our vetting process. The control group trained the data without vetting. Fig. 3 is a box and whisker representation of the spread of the accuracy scores across for each model. From these results, it shows the accuracy we get from predicting with the vetted data is better.

V. CONCLUSION

In this paper, we devised an approach that attempts to mitigate the data poisoning issue in a federated learning network. In our approach, we introduce the concept of a facilitator that gets assigned to an end device. The facilitator's job is to ensure the data that an end device owns has not been compromised. It achieves this by employing an SVM model for the data vetting process. We run experiments to determine the effectiveness of our approach. Our experiment showed that a model's accuracy is better when the data it trains on has been positively vetted.

REFERENCES

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[2] S. Safavat, N. N. Sapavath, and D. B. Rawat, "Recent advances in mobile edge computing and content caching," *Digital Communications and Networks*, vol. 6, no. 2, pp. 189–194, 2020.

[3] M. Al-Alshaqi and D. B. Rawat, "Cloud, edge, and fog computing and security for the internet of things," *EAI Endorsed Transactions on Internet of Things*, vol. 6, no. 23, 2020.

[4] R. Doku and D. B. Rawat, "IFLBC: On the Edge Intelligence Using Federated Learning Blockchain Network," in *2020 IEEE Intl Conference on Intelligent Data and Security (IDS)*, pp. 221–226, 2020.

[5] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[6] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *arXiv preprint arXiv:1909.11875*, 2019.

[7] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[8] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[9] B. Han, I. W. Tsang, and L. Chen, "On the convergence of a family of robust losses for stochastic gradient descent," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 665–680, Springer, 2016.

[10] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar, "Antidote: understanding and defending against poisoning of anomaly detectors," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pp. 1–14, 2009.

[11] A. D. Joseph, P. Laskov, F. Roli, J. D. Tygar, and B. Nelson, "Machine learning methods for computer security," *In: Dagstuhl Manifestos, Dagstuhl Perspectives Workshop*, vol. 1237, 2013.

[12] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[13] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?," in *International Conference on Machine Learning*, pp. 1689–1698, 2015.

[14] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter.," *LEET*, vol. 8, pp. 1–9, 2008.

[15] A. Paudice, L. Muñoz-González, A. Gyorgy, and E. C. Lupu, "Detection of adversarial training examples in poisoning attacks through anomaly detection," *arXiv preprint arXiv:1802.03041*, 2018.

[16] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[18] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," in *Advances in neural information processing systems*, pp. 3517–3529, 2017.

[19] S. Bauer and N. B. Priyantha, "Secure data deletion for linux file systems.," in *Usenix Security Symposium*, vol. 174, 2001.

[20] P. Gutmann, "Secure deletion of data from magnetic and solid-state memory," in *Proceedings of the Sixth USENIX Security Symposium, San Jose, CA*, vol. 14, pp. 77–89, 1996.

[21] R. Doku, D. B. Rawat, and C. Liu, "Towards federated learning approach to determine data relevance in big data," in *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 184–192, 2019.