Reducing communication and computation cost by generating datasets from decision trees in a federated learning environment

Thilo Bartels
thilo.bartels@stud.uni-hannover.de
Leibniz University Hannover

Johann Bartels
johann.ulrich.bartels@stud.uni-hannover.de
Leibniz University Hannover

Abstract

Federated Learning in IoT environments is limited by computational limits on the clients and the communication ability between the clients and the server. In this work we propose a method that uses Decision Trees as client-models which are send to the server in order to generate a synthetic dataset. That dataset is then used to train a global model that unites the decisions of the client-models. Compared to Fed-Avg and other federated learning approaches we are using smaller, easier to compute models and just one communication round and are thereby reducing communication and computation cost while also being able to produce an adequate global-model.
Our code is available at[1] [https://github.com/Bartels-2/Federated-Learning-with-Decision-Trees].

1 Introduction

Introduction: Federated Learning enables model training across multiple clients while preserving data privacy. When applied in an IoT environment the training can be restricted by the clients capacity concerning the communication and the computation abilities. Therefore one may have to find a compromise between the communication cost, the computation cost and the overall model performance of the applied federated learning method.

Background Literature (Related Works): Existing work like High-Compression-Federated-Learning (HCFL)[4] has focused on reducing the communication cost of Federated learning through compression of neural networks with the help of autoencoders while otherwise using the standard FedAVG [3] approach for weight averaging . Other work like Fed-Trees[1] uses tree ensamble models to reduce both communication and computation cost.
Motivation: Problems with works like HCFL are that we are still dependent on compuationally demanding neural networks while also adding the auto-encoder, which has to be computed on the client. Other methods like Tree-Fed depend on tree ensamble models which are computationally less demanding and smaller than neural networks but still require a comparatively high computation cost compared to simpler models like decision trees. This method additionally also is dependent on a validation dataset on the server in order to select the best client model to distribute in the next training round.
Contribution: In our work, we tackle these issues by introducing a novel technique that improves the client-server communication cost and the computation cost on the clients. The main contributions of our proposed work are summarized below: ***(i).*** We present our method to generate a synthetic dataset based on decision tree models ***(ii).*** We apply our method to two benchmark datasets and show that we reduce the communication and computation cost compared to our baseline method Fed-AVG and also show the overall performance of our resulting global model.

2 Proposed Methodology

Our Federated Learning method can be divided into two sides, the client-side and the server-side.
Client-Side: On the client side each client trains one decision tree model on its local dataset and shares it with the server after training. The decision tree uses the entropy of nodes as loss criterion in order to achieve purer nodes.
Server-Side: On the server side the server receives one decision tree model per client. These are then used to generate a synthetic dataset on which a global model can be trained. The overall goal of the process is to get a synthetic dataset that represents the most important decisions of all client-models. Thereby the dataset can be used to train a global model that aggregates the client-models. For the generation process each path from leaf-node to root-node is used to generate data that fits to the respective path. To sample the values of the datapoints the split decisions are taken into consideration and features that are not used in the current path are set to a value outside of the range of values. The classes of the synthetic-dataset are generated through a majority vote of the client-models. That means that all client-models predict a class for each datapoint and the most often predicted class gets assigned.

In order to better represent the client-models we also use sub-paths of the path's between leafs and roots to generate the synthetic data. These sub-paths start at the leaf and do not take nodes that were cut from the original path into consideration during the generation process. Based on the length of the sub-path in respect to the full-path we are then adding entropy to the datapoints that were generated from sub-paths. This is done by flipping the assigned class of the datapoints after they were assigned via majority vote. Here the probability of each flip depends on the path-length where datapoints that were generated by shorter sub-paths have a higher possibility to be flipped than datapoints that were generated by longer sub-paths. The idea of adding entropy in order to generate synthetic-data that can be used to form the decision-tree, from which the data was generated from, was proposed by [5] and

[1] https://github.com/Bartels-2/Federated-Learning-with-Decision-Trees

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

adapted for our federated-learning environment. It leads to the result that the synthetic data which is generated by one decision tree represents the decisions of the decision tree while also reflecting the hierarchy of the decisions.

In Algorithm 1 one can see the core part of our method. Here we provide the client-models as input and get the synthetic X values and a list of datapoints where the class will be flipped as outputs.

Algorithm 1 Generate synthetic Data

Initialize empty lists: X_syn_result, $y\_syn\_flipped\_result$
for each tree in $trees$ **do**
  $paths \leftarrow$ traverse_tree($tree$)
  for each ($path, length$) in $paths$ **do**
    for $i \leftarrow 1$ to $length$ **do**
      Extract last i conditions in $path$
      Initialize $first\_run \leftarrow 1$
      for each $condition$ in $path[-i:]$ **do**
        $feature, operator, threshold \leftarrow condition$
        if $operator$ is $\leq$ **then**
          $split\_feature\_values \sim \mathcal{U}(threshold - \alpha \cdot threshold, threshold)$
        else if $operator$ is $>$ **then**
          $split\_feature\_values \sim \mathcal{U}(threshold, threshold + \alpha \cdot threshold)$
        end if
        if $first\_run = 1$ **then**
          Initialize $X\_syn \sim [m, n]$
          Initialize $y\_flipped \leftarrow 0$
          $first\_run \leftarrow 0$
        end if
        $X\_syn[feature] \leftarrow split\_feature\_values$
      end for
      if $i < length$ **then**
        For each label $y_i \in y\_flipped$, sample y_i as:

$$y_i = \begin{cases} 1 & \text{with probability } p, \\ 0 & \text{with probability } (1-p), \end{cases}$$

        where $p = f(\text{length})$
      end if
      Append X_syn, $y\_flipped$ to X_syn_result, $y\_syn\_flipped\_result$
    end for
  end for
end for
return $X\_syn\_result, y\_syn\_flipped\_result$

3 Experiments

3.1 Datasets

- The IoT dataset consists of 37580 samples with 17 features and was taken from [6]. The task for the dataset is to classify whether an air condition unit of an hotel is broken or not. For our experiment we split the dataset into 75% training data, 25% test data and distributeed the training data to our clients.
- The census income dataset consists of 48842 samples with 14 features and is taken from [2]. The task for the dataset is to predict whether the annual income of a person is above 50 thousand. For our experiment we cleaned the dataset from samples with missing values and used a one-hot encoding for the features. The data was then split up into 1/3 test data and 2/3 training data which was distributed to the clients.

3.2 Implementation Details

Hyperparameters client-side Maximal depth for decision trees: 3; training criterion: entropy loss; number of clients: 200
Hyperparameters server-side Maximal depth of decision trees: 4 for IoT-dataset, 7 for census-dataset;training criterion: entropy loss; number of synthetic samples per path: 100; $\alpha = 0.1$; $m = -100000$; $n = 500000$; p for sub-path with one missing node = 0.55; p for sub-path with two missing nodes = 0.65;
Hyperparameters Fed-Avg (Benchmark) Number of clients: 200; hidden-layer size [12,12]; learning-rate: 0.01; number of epochs per round 10:
Evaluation Metrics: For our evaluation we performed 15 independent runs of our decision tree based method and of Fed-AVG while keeping the splits for the client the same for two corresponding runs. The Fed-AVG algorithm within our experiment was set to run until beating the test accuracy score of the respective decision tree model.
Environment Details: The implementation was written in Python 3.12 and executed within a Visual Studio Code environment. Since the experiments were not conducted in a dedicated standalone setup or cloud environment, all computations were run locally on a single machine.
Computation Usage: All experiments were conducted on a system equipped with an AMD Ryzen 7 5800U processor (8 cores, 16 threads) and 16GB RAM. No external GPU was used, and all computations were performed on the CPU. Details regarding the computational time for each experiment can be found in the **Results** section.

3.3 Baselines

4 Results

In this section, we review the results of our methodology by comparing the performance of our decision tree-based approach with a Fed-AVG approach using neural networks. The analysis combines direct observations and relevant calculations to offer practical insights.

4.1 Decision Tree Metrics

Table 1 shows how our decision tree-based approach performed on the IOT and census datasets. Key findings include:

- **Final Training and Test Accuracy:** The decision tree achieved training accuracies of 63.58% (IOT) and 65.55% (census). Test accuracies were 85.62% (IOT) and 75.43% (census), suggesting that the IOT dataset allowed the model to generalize better. Differences in data complexity and

Metric	IOT Dataset	Census Dataset
Final Training Accuracy(%)	0.6358	0.6555
Final Test Accuracy (%)	0.8562	0.7543
Tree Size (Byte)	48	48
Avg. Computation Time (sec)	0.0012	0.0010

Table 1: Comparison of Decision Tree metrics for the IOT and Census dataset.

Metric	IOT Dataset	Census Dataset
Final Test Accuracy (%)	0.8440	0,7543
Rounds	34.4666	1
Avg. Weight Size (Byte)	480	480
Avg. Comp. Time per Epoch (sec)	0.0011	0.0013
Avg. Comp. Time per Round (sec)	0.0112	0.0138

Table 2: Comparison of Neural Network metrics for the IOT and Census dataset.

distribution likely caused these variations. The comparatively low training accuracy can be explained by the entropy added during the data generation process, which results in the model being unable to learn split decisions that separate the classes in the synthetic dataset in a better way.

- **Communication and computation efficiency:** Each decision tree used 48 bytes of memory, which can be attributed to clients having the same hyperparameters for the decision trees. Since our approach only uses one one-way communication round, each client only had to send the model once. Computation times for each client were 0.0012 seconds for the IOT dataset and 0.0010 seconds for the census dataset.

4.2 Neural Network Metrics

Table 2 shows how the neural network-based Fed-AVG approach performed across the datasets. Below is an analysis of these results:

- **Final Training and Test Accuracy:** The neural network achieved a test accuracy of 84.40% for the IOT dataset and 75.43% for the census dataset over an average of 34.47 rounds for IOT and 1 round for census.
- **Communication efficiency:** Although the average weight size was 480 bytes per client, this size effectively doubles per communication round as the model is sent to the server and then back to the client. Over 34.47 rounds (IOT dataset), the cumulative communication load adds up to $480 \times 2 \times 34.47 = 33,091.2$ bytes exchanged per client. For the census dataset, $480 \times 2 \times 1 = 960$ bytes per client, resulting in a much lower communication cost.
- **Computation efficiency:** The neural network required 34.47 rounds to achieve the same test accuracy as the decision tree on the IOT dataset, significantly more than the 1 round needed for the census dataset. When combined with the average computation time per round (0.0112 seconds for IOT and 0.0138 seconds for census), the total computation time for convergence can be estimated as follows:
  - **IOT Dataset:** Total computation time = $34.47 \times 0.0112 \approx 0.3861$ seconds.
  - **Census Dataset:** Total computation time = $1 \times 0.0138 = 0.0138$ seconds.

  These results illustrate that while the IOT dataset required more rounds, the overall computational cost remained manageable due to shorter computation times per round.

4.3 Comparative Insights

The results highlight a clear trade-off between decision trees and neural networks in federated learning:

- **Accuracy vs. Efficiency:** Fed-AVG was able to achieve slightly higher test accuracies than our decision tree-based approach. For example, the IOT dataset test accuracy was 84.40% (NN) compared to 85.62% (DT), and for the census dataset, 75.43% (NN) versus 75.43% (DT).
- **Computation costs:** The decision trees required much less computation time per client (0.0012 seconds IOT, 0.0010 seconds census) compared to the significantly higher computation times required for the neural networks (0.3861 seconds IOT over 34.47 rounds, 0.0138 seconds census over 1 round). This results in a faster computation by a factor of $\frac{0.3861}{0.0012} \approx 321.75$ for the IOT dataset and by a factor of $\frac{0.0138}{0.0010} \approx 13.8$ for the census dataset.
- **Communication Costs:** Decision trees require minimal communication due to their smaller size and single-round updates. Neural networks, in contrast, involve much higher communication overhead, especially for datasets requiring many rounds. For our experiment with the IOT dataset, Fed-AVG used 34.47 rounds, resulting in a $\frac{33,091.2}{48} \approx 689.4$ times greater communication cost than the single-round decision tree process. For the census dataset, the communication cost was $\frac{960}{48} = 20$ times greater.
- **Scalability and Resource Constraints:** Decision trees are ideal for resource-constrained settings, such as edge devices or low-bandwidth environments like IoT. Neural networks (Fed-AVG), while more demanding, are better suited for scenarios where accuracy is critical, and resources like bandwidth and processing power are more readily available.
- **Convergence Dynamics:** The higher number of rounds required for Fed-AVG (e.g., 34.47 for IOT) reflects its greater complexity. However, this can also allow it to extract more meaningful patterns from data. Decision trees, with their single-round updates, provide quicker but possibly less detailed insights.

4.4 Conclusion

In our work, we proposed a method for federated learning in IoT environments, targeting the problems of computation cost and communication cost. Our experiments showed that our approach was able to reduce the computation cost by a factor of 321.75 for the IoT dataset and by a factor of 13.8 for the census dataset while also reducing the communication cost by a factor of 689.4 for the IoT dataset and a factor of 20 for the census dataset, when compared to a basic Fed-AVG approach. This was achieved while only reducing the model's prediction accuracy by a few percent, which makes our

approach viable in IoT environments where communication and computation abilities are limited, and where the task complexity or demanded model performance is appropriate.

References

[1] Mohammad Al-Quraan, Ahsan Khan, Anthony Centeno, Ahmed Zoha, Muhammad Ali Imran, and Lina Mohjazi. 2022. Fedtrees: A novel computation-communication efficient federated learning framework investigated in smart grids. *arXiv preprint arXiv:2210.00060* (2022).

[2] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

[3] 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.

[4] Minh-Duong Nguyen, Sang-Min Lee, Quoc-Viet Pham, Dinh Thai Hoang, Diep N Nguyen, and Won-Joo Hwang. 2022. HCFL: A high compression approach for communication-efficient federated learning in very large scale IoT networks. *IEEE Transactions on Mobile Computing* 22, 11 (2022), 6495–6507.

[5] Taoxin Peng and Florian Hanke. 2016. Towards a synthetic data generator for matching decision trees. In *International Conference on Enterprise Information Systems*, Vol. 2. SCITEPRESS, 135–141.

[6] Raneen Younis and Marco Fisichella. 2022. FLY-SMOTE: Re-Balancing the Non-IID IoT Edge Devices Data in Federated Learning System. *IEEE Access* 10 (2022), 65092–65102. https://doi.org/10.1109/ACCESS.2022.3184309

A Research Methods

For our research we ran quantitative experiments from which we extracted relevant metrics like model size, computation time and model accuracy in order to evaluate our method and to compare it with a standard Fed-AVG approach. The experiment has been conducted 15 times and the results have been averaged in order to account for variability.

B Online Resources

Our code is available at[2] [https://github.com/Bartels-2/Federated-Learning-with-Decision-Trees].

Limitations

Datasets: Our research demonstrates the possible performance of our method on two different datasets. While our method was able to reduce both the communication costs and the computation cost in both instances, when compared to a standard Fed-AVG algorithm, the factor of improvement differed significantly for the datasets. Further research could test our methods performance on further datasets in order to investigate the relationship between the datasets characteristics and the methods performance.

Architecture: The experiments for which the data was collected used a fixed architecture for each dataset. Additional research could further look into the relationships between different architectures, the computation costs, the communication costs and the model performances.

Tasks: In our work we applied our method to a binary classification problem. Applying it to multi-class or regression problems might require further adaptations to the data generation process.

Number of clients: Our experiments only tested the methods on a fixed number of clients (200). Further research could investigate the advantages and disadvantages of our method for smaller and larger amounts of individual client systems.

[2] https://github.com/Bartels-2/Federated-Learning-with-Decision-Trees