Analysis Report

Ghazal Ebrahimi

Overview

This document explores the non-IID data challenge in federated learning by following the steps outlined in the assignment. It begins with an explanation of the problem, followed by detailed descriptions of the implementation for each task. The final section presents and analyzes the experimental results. All source code is available on GitHub, along with a `.yml` environment file and a `README.md` containing setup instructions and usage guidelines.

Introduction

Federated learning is a machine learning paradigm where multiple clients collaboratively train a model using their own local datasets, without sharing raw data. Most federated learning algorithms assume that data is independent and identically distributed (IID) across clients. However, in real-world scenarios, this assumption rarely holds. In practice, client datasets are often non-IID, meaning they are not drawn from the same distribution and may not be representative of the overall data distribution.

The non-IID data problem is one of the most significant challenges in federated learning, as it can greatly reduce the performance of global models. One of the foundational algorithms in this field is Federated Averaging (FedAvg), introduced by McMahan et al. in their paper *"Communication-Efficient Learning of Deep Networks from Decentralized Data."* FedAvg works by averaging the parameters of locally trained client models, with each client's contribution weighted by the size of its dataset. While this method is effective in handling quantity skew (when clients have different amounts of data), it performs poorly under label distribution skew, a common form of non-IID data where the distribution of class labels varies significantly across clients.

In this assignment, we simulate a federated learning setup with two clients using the MNIST training dataset. We create two subsets of the dataset with imbalanced label distributions to represent a scenario of high label distribution skew. A custom model is trained independently on each client's subset, and then a simplified version of the FedAvg algorithm is applied by averaging the saved model weights.

To address the limitations of standard FedAvg under label imbalance, we propose and implement two modifications to the aggregation method. Finally, we evaluate all trained models, present the experimental results, compare model performances, and provide a comprehensive analysis of the impact of label skew and aggregation strategy on model effectiveness.

Implementation

I first downloaded and saved the MNIST training dataset. Then, I created a highly skewed non-IID split of the dataset into two clients:

- Client 1 will have 80% of digits 0-4 and 20% of digits 5-9
- Client 2 will have 20% of digits 0-4 and 80% of digits 5-9

To do that, I iterated over each class, and using the given weights for class distribution, I randomly selected a subset of data points for each class to assign to one client. The remaining data points from that class were then assigned to the other client. This way, I ensured that each class was split between the clients according to the specified proportions, resulting in a controlled label-skewed distribution across the two clients. Figure 1 shows the data distribution across classes for each resulting split.

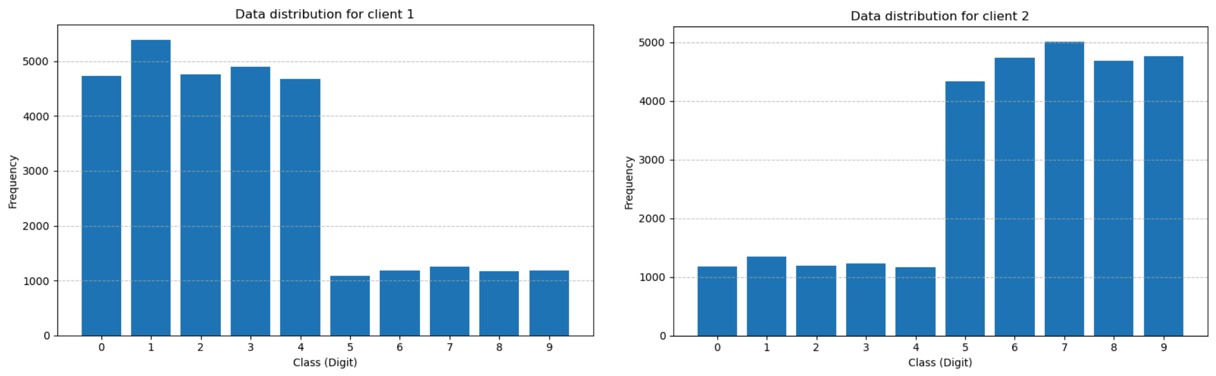

Figure 2. Clients datasets class distributions

For the next step, I implemented a simple CNN model consisting of one convolutional layer followed by a pooling layer and two fully connected layers, as shown in the below code snippet:

```
self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3) # 1 × 28 × 28 ->
16 x 28 x 28
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2) # -> 16 x 13 x 13
        self.fc1 = nn.Linear(2704, 128) # 2704 -> 128
        self.fc2 = nn.Linear(128, 10)  # -> 10
```

I then implemented a training script that accepts a client ID as an input argument and trains an instance of the classifier on the corresponding client-specific data split prepared in the previous step. I used the Adam optimizer and the cross entropy loss for training. Models are trained for 15 epochs using a batch size of 64, and the resulting model parameters are saved in corresponding `.pth` files. Using this script, I trained the model separately on both client datasets and saved the learned parameters. Additionally, I trained the same model architecture on the original MNIST training dataset without any label skew or client split. This model serves as a baseline for comparison and is referred to as the *base model* throughout the rest of the document. The plots in Figure 2 illustrate loss and accuracy curves for clients 1 and 2 during training.

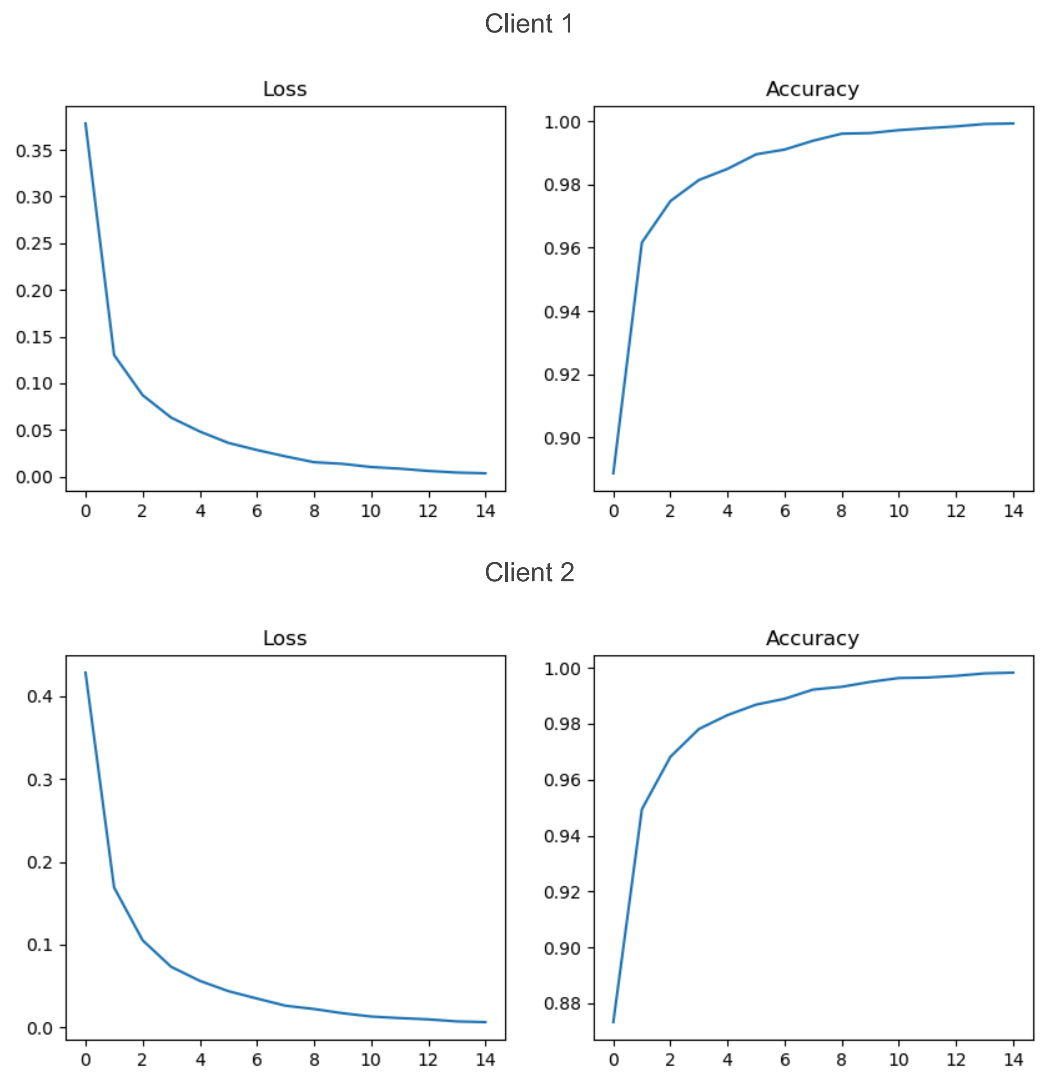

Figure 2. Loss and accuracy curves for training over the number of epochs

Next, I implemented a simplified version of the standard FedAVG algorithm based on the following aggregation rule provided in the document:

$$\theta_{t+1} = (n_1/n) \text{ x } \theta_1^{t} + (n_2/n) \text{ x } \theta_2^{t} + \ldots + (n_k/n) \text{ x } \theta_k^{t}$$

Since this setup involves only 2 clients with an equal number of data points, I simplified the weighted averaging to:

$$\theta = 0.5 \text{ x } \theta_1 + 0.5 \text{ x } \theta_2$$

I loaded the trained model parameters from each client, verified parameter compatibility, and performed element-wise averaging across all parameters. The resulting global model parameters were saved in a separate file for later evaluation.

To specifically address the label distribution skew challenge in this assignment and potentially improve the global model's performance, I explored two modified versions of the FedAvg algorithm:

Approach 1: Averaging All Layers Except Classifier

Instead of averaging the entire model, I averaged only the feature extraction layers and selected the classifier layer from one of the clients. This avoids degrading class-specific representations learned by each client.

```
for key in state_dict1:
       if "fc2" in key:
           # pick from better client
           averaged_state_dict[key] = state_dict1[key].clone()
       else:
           averaged_state_dict[key] = 0.5 * state_dict1[key] + 0.5 * state_dict2[key]
```

Approach 2: Class-wise Weighted Averaging for Classifier

For the classifier layer, I used a custom averaging strategy where the weights for each class output are averaged, with more weight given to the client that saw that class more frequently. For instance, weights for classes 0–4 were biased toward Client 1, and weights for classes 5–9 were biased toward Client 2.

```
for i in range(weight.size(0)):  # 10 output classes
       if i < split_index:
           w = w_client1  # more trust in client 1
       else:
           w = 1 - w_client1  # more trust in client 2

       weight[i] = w * fc1.weight[i] + (1 - w) * fc2.weight[i]
       bias[i] = w * fc1.bias[i] + (1 - w) * fc2.bias[i]
```

In the following section, the experimental results and model evaluations are presented, followed by a discussion and comparison of the different approaches implemented in this assignment.

Results and Analysis

In this section, the experimental results based on previous steps are presented. Tables 1 and 2 show the classification accuracy of each client model on their dataset and the other dataset, respectively. Table 3 presents all models' accuracies on the shared MNIST test dataset. Following the tables, the confusion matrices are presented, illustrating the performance of client models and the standard FedAvg model on shared test data. We will go through all the results and analyze them in the rest of the document.

As presented in Table 1, client 1 and client 2 each perform very well on their respective training split. According to Table 2, the performances slightly drop (around 3%) on the other training split but are still fairly good. Looking at Table 3, it can be said that despite the label imbalance in their training data, both local models generalize reasonably well to the global test dataset, showcasing a high accuracy on all classes and in total but slightly less than the accuracy on their training data. Therefore, as expected, each model performs better on its training dataset compared to the other training split and the global shared test set.

In all experiments, the locally trained models demonstrate higher accuracy on the classes that are over-represented in their respective training datasets. Specifically, Client 1 performs better on digits 0–4, while Client 2 achieves higher accuracy on digits 5–9, reflecting the label bias in their local data distributions. This pattern is more clearly observed in Tables 2 and 3, where each model is evaluated on the other client's dataset. In contrast, Table 1 shows near-perfect performance on the models' own data, with subtle differences that are less pronounced due to the models being highly optimized for their respective distributions.

Table 1. Accuracy on own training split

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
Client 1	1.0000	0.9998	0.9998	0.9996	1.0000	0.9982	0.9992	1.0000	0.9991	0.9941	**0.9995**
Client 2	0.9983	1.0000	1.0000	0.9992	0.9991	0.9991	1.0000	0.9970	0.9998	0.9994	**0.9991**

Table 2. Accuracy on the other client’s training split

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
Client 1	0.9992	0.9970	0.9857	0.9886	0.9932	0.9675	0.9755	0.9787	0.9483	0.9372	**0.9680**
Client 2	0.9648	0.9794	0.9576	0.9568	0.9626	0.9815	0.9941	0.9840	0.9906	0.9874	**0.9690**

Table 3. Accuracy on global test set

Class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
Base Model	0.9959	0.9965	0.9816	0.9861	0.9959	0.9832	0.9843	0.9903	0.9795	0.9703	**0.9865**
Client 1	0.9980	0.9991	0.9845	0.9931	0.9969	0.9742	0.9687	0.9786	0.9548	0.9316	**0.9783**
Client 2	0.9714	0.9797	0.9564	0.9673	0.9664	0.9865	0.9916	0.9874	0.9867	0.9822	**0.9774**
Standard FedAvg	0.8908	0.9665	0.8091	0.4990	0.9450	0.6839	0.9687	0.8016	0.9815	0.6977	**0.8259**
Modified FedAvg 1	0.9847	0.9894	0.9409	0.8921	0.9776	0.9585	0.9843	0.9971	0.9846	0.9415	**0.9652**
Modified FedAvg 2	0.9908	0.9938	0.9738	0.9792	0.9919	0.9776	0.9708	0.9815	0.9733	0.9643	**0.9799**

On the other hand, the global model obtained through standard FedAvg demonstrates relatively poor performance on the shared test set. This is because FedAvg simply averages the parameters of the local models without accounting for the label imbalance. As a result, it produces a global model that fails to specialize in either class subset.

Figures 3 and 4 illustrate the confusion matrices for the client models and the standard FedAvg model, respectively, on the shared test dataset. It can be seen that the same interpretations can be drawn from these matrices as from the accuracy tables. Both local models show strong performance, especially on the classes that are over-represented in their respective training distributions. In contrast, the FedAvg model demonstrates significantly worse performance, with more frequent misclassifications across multiple classes. This visual representation further confirms that the standard FedAvg model struggles to generalize well in the presence of label distribution skew, supporting the quantitative results.

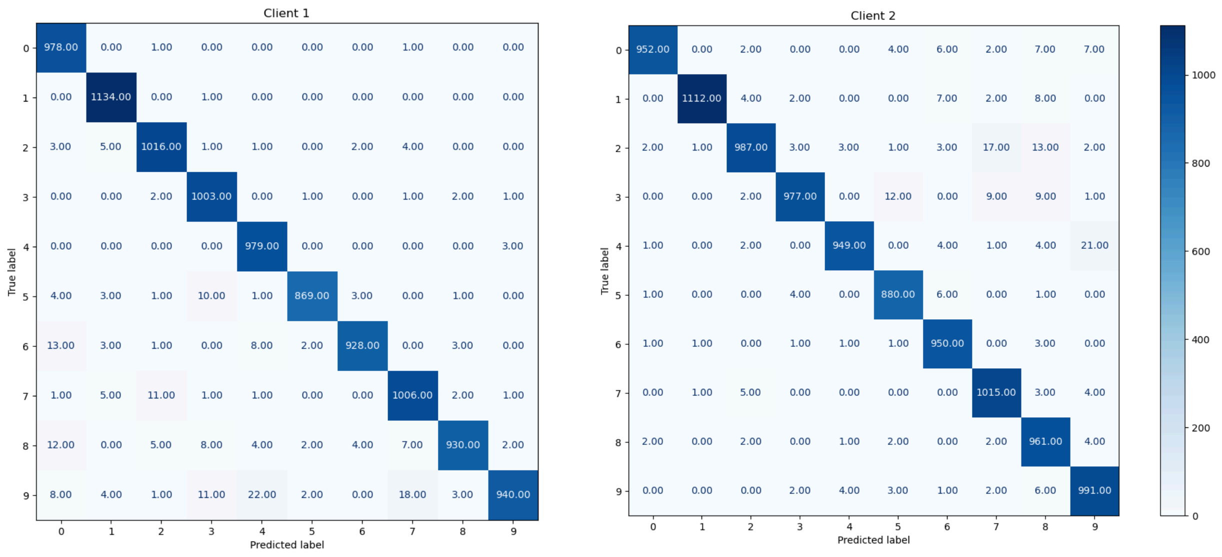

Figure 3. Confusion Matrices of locally trained models on shared test dataset

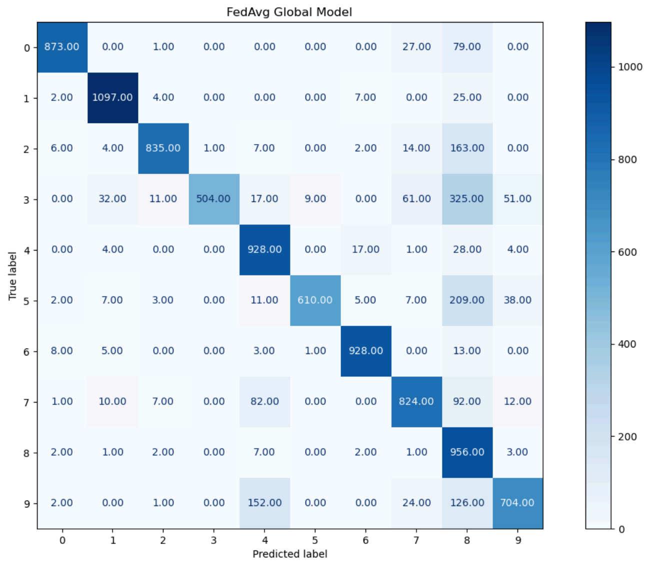

Figure 4. Confusion Matrix of FedAvg global model on shared test dataset

This issue arises from the problem of divergent local objectives. FedAvg assumes that local updates are noisy approximations of the same global optimum; however, this assumption breaks down under non-IID label distributions. Client 1 becomes highly specialized in digits 0–4, while Client 2 focuses on digits 5–9. Averaging these models leads to a compromise that performs poorly across the full class range. This phenomenon, known as client drift, occurs because client updates point in different directions in parameter space, leading to a global model that underperforms relative to the individual local models.

Without modifications, FedAvg struggles as heterogeneity increases. While simple and efficient on IID data, label skew can introduce bias, slow convergence, and low accuracy.

The last two rows of table 3 show the evaluation accuracy of the modified FedAvg models, explained in the previous section. As you can see, both modifications significantly improve the accuracy across all classes and in total compared to the standard FedAvg.

In the first approach, the classifier layer of the global model adopts the parameters of the best client model. For the other parameters, the method follows the standard FedAvg approach, taking the average of the clients' parameters. This approach relies on the fact that the classifier layer is the most sensitive to label distribution skew because it directly maps features to class predictions. When each client sees mostly different classes, their classifier layers become biased toward those classes. Averaging them destroys both clients' specialized knowledge. By only averaging the feature extraction layers and keeping or selecting the classifier layer from one client, we avoid mixing incompatible outputs and preserve effective class mappings.

In the second approach, instead of averaging all classifier weights equally, this method gives more influence to the client that had more examples for each class. Since Client 1 is better at digits 0–4 and Client 2 is better at digits 5–9, soft-averaging lets clients contribute more where they are strong. This preserves useful class-specific representations and leads to a more balanced and accurate classifier overall. According to the experiments, this approach achieves the most accurate model, outperforming client 1, client 2, and the other two aggregated models on MNIST test dataset, with a classification accuracy of ***97.99*** %.

Conclusion

Label distribution skew is a critical challenge that can severely impact the performance of federated models if left unaddressed; however, there is a growing set of techniques designed to mitigate its effects. By applying the appropriate method or a combination of methods, the performance of FedAvg on non-IID label distributions can be significantly improved. This helps bring the global model closer to the IID baseline, resulting in more reliable and fair performance across all clients. In conclusion, while FedAvg remains the baseline algorithm for federated learning, it performs poorly in the presence of label distribution skew, as it overlooks the differences in the underlying data distributions across clients. Each client is optimized for its own skewed distribution, and simply averaging their model parameters does not reflect the true global distribution, which leads to degraded performance.