COMP3221 Assignment2 Report

Federate Learning

SID: 520627482

SID: 510356677

Abstract—**In this assignment, we have implemented a federate learning system that consists of one server and five clients. Each client possesses its own private dataset used for training a local model, and then contributes its local model to the server in order to build a global model. The main focus are:**

1. Scale machine learning across multiple devices while ensuring data privacy and minimizing central data storage needs.

2. Network communications using sockets, including setting up connections, designing protocols, and handling network issues.

3. Implement, train, and evaluate machine learning models, focusing on practical aspects such as data handling, model and performance optimization.

I. Analysis

A. Dataset

We use the California Housing Dataset [4], which is a famous dataset used in early machine learning projects.

The expected output is the value of the median house value for different blocks in California. The input will be 8 features: median income, housing median age, average rooms, average bedrooms, population, average occupancy, latitude, and longitude.

The dataset is already split and allocated to five clients. Each client receives a portion of the dataset, varying in size. And the received data is divided into training set and testing set.

B. Model

We utilize the classic single-layer Linear model for our Linear Regression Model which models the correlation between house features and their prices. While we considered incorporating non-linear layers or using multiple Linear layers, doing so would violate the assignment's requirement for a **Linear Regression** Model.

Therefore, without altering the simplicity of the Linear Regression Model, we incorporated normalization techniques [1] and the ADAM optimizer [3] to enhance the model's performance.

C. Loss Function

We choose Mean Squared Error (MSE) as our loss function because it best suits the regression task. In the experimental section, we also employ MSE as our evaluation metric.

II. Implementation

A. Handshake

Initially, the server starts up, initializes values, and awaits the client's handshake request.

Then upon client starts, it sends a handshake request to port 6000, the address of the server.

Once the server receives it, it adds the client to its stored list of clients, records its ID, dataset size, and so on. Then, it replies to the client. Upon receiving the reply, the client prepares to receive the server's first round of models.

Upon receiving the first handshake, the server will wait for 30 seconds then start distributing the model.

B. Server Distribute

In the first round, the server randomly initializes model parameters and distributes them to each client. Then, it waits for each client to reply with its own model.

C. Local Testing

Each time a model is received from the server, the client performs testing first. We set the batch size of the test dataloader to 1 to simulate how the model is typically used in practice, with users inputting data one by one.

D. Local Update

The client determines whether to use Batch Gradient Descent or mini-batch Gradient Descent based on the parameters passed at startup. In every epoch, if using GD, the client will take all observations allocated to it and train. Otherwise using mini-batch GD, the client will randomly pick some and train.

It determines the number of epochs to train based on the hyperparameter num_epochs then returns the trained model to the server.

E. Server Aggregation

The server maintains a variable to track whose models it has received. If all registered clients have sent their models, the server aggregates (by averaging) them and redistributes the model parameters.

F. Training End

Training will conclude as soon as the number of global iteration reaches the number we set, at which point a 'finished' message will be sent to clients to close their sockets.

G. New Client Joining

All clients who join during the registration period will participate in training from the beginning. Any new client that joins during an iteration will be flagged as not ready. Once current iteration is complete, they will be marked as ready and will join the next iteration.

III. Additional

A. Normalization

When the client startup, the client will perform a normalization [1] on the dataset allocated to it.

Based on our analysis, the data points received by each client are highly scattered. Therefore, we performed normalization to reduce and smooth out the standard deviation of the data distribution.

We analyzed the data points and normalized them to a reasonable range based on their original mean and variance. For example, we set the range of median income to (0, 15) and normalized all values to this range, thus eliminating excessively large outlier values (By normalizing them to a reasonable range, while also ensuring the consistency in the data's relative sizes).

We are unable to provide the distribution plot drawn here because of the page limit (One plot for each parameter, 8 plots drawn totally, too large to put within 3 pages)

B. ADAM

We have utilized ADAM [3] as our optimizer. ADAM, short for Adaptive Moment Estimation, incorporates momentum and adaptive learning rates in addition to stochastic gradient descent (SGD). This contributes to smoother and faster convergence.

C. Client failure

Upon one client fails, it won't send its local model to the server. So if the server hasn't received the model from one client for 30 seconds, it will conclude this client has failed and terminate the whole federate learning process (by sending 'finished' to all clients).

IV. Experimentation

A. Experimental Setting

According to the assignment requirements, what we actually did was not the traditional evaluation (comparing with other models), but rather parameter tuning. We used grid search [2] for experiments, where we focused on one hyperparameter at a time, while keeping other hyperparameters fixed, to find the optimal value for the currently focused hyperparameter. Then, we fixed this optimal value and tested other hyperparameters.

B. Evaluation on learning rate

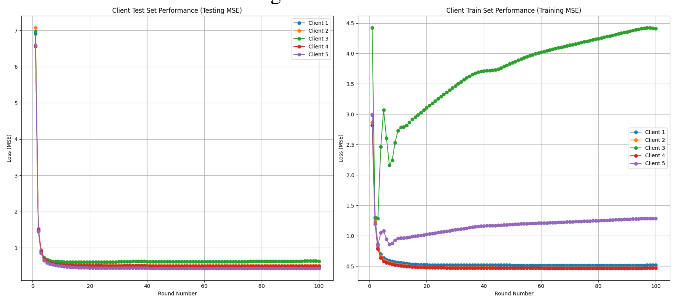
We have fixed to use Gradient Descent, set number of epochs to 10, disable subsampling, then evaluate on various learning rates for 100 global iterations. The chosen values are 0.001, 0.002, 0.005, 0.01, 0.05, 0.1, 1.0. See table I

It is clear that when the learning rate increases, the "number of iterations when convergence" decreases and the loss also decreases (Because with a larger learning rate, the convergence will be faster). lr=1.0 seem to be the best choice. **However**, based on the plots 1 generated by our program, we observed that the model's robustness is poor when lr=1.0. It's evident one client's training loss exhibits significant fluctuations. Similar phenomena were observed when lr=0.1. Therefore, we opted for a more stable lr=0.05, as depicted in the following graph 2.

TABLE I
EVALUATION ON LEARNING RATE

Learning rate	Minimum MSE	number of iterations when convergence
0.001	1.236	above 100
0.002	1.199	above 100
0.005	0.993	above 100
0.01	0.675	80
0.05	0.535	40
0.1	0.515	15
1.0	0.503	5

Fig. 1. Plot: lr=1.0



C. Evaluation on number of epochs

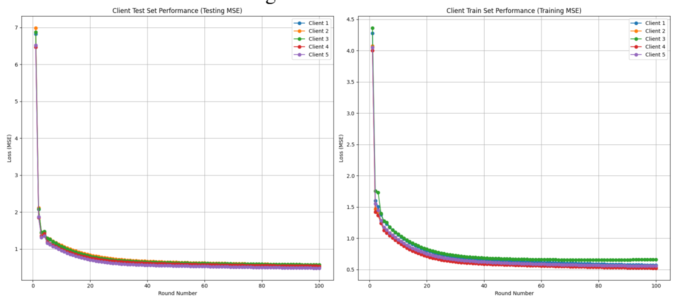
We have fixed to use Gradient Descent, set learning rate to 0.05, disable subsampling, then evaluate on various values chosen for number of epochs. The chosen values are 5, 10, 15, 20. See table II

TABLE II
EVALUATION ON NUMBER OF EPOCHES

Number of epochs	Minimum MSE	#iterations when convergence
5	0.593	80
10	0.535	40
15	0.509	30
20	0.506	20

When the number of epochs increases, the "number of iterations until convergence" decreases, and the loss also decreases.

Fig. 2. Plot: lr=0.05



We assume this is because, with more epochs trained for each global iteration, the clients learn more patterns among the data. Clearly, within the same global iteration, the more epochs each iteration uses, the more total epochs are trained.

We optimized the number of epochs to 20 as it achieves the best performance.

D. Evaluation on GD vs mini-batch GD with different batch sizes

We have fixed to set learning rate to 0.05, set number of epochs to 20, and disable subsampling, then evaluate on GD or mini-batch GD with various chosen batch size. The chosen values are 1, 150, 300, 750, 1500. See table III

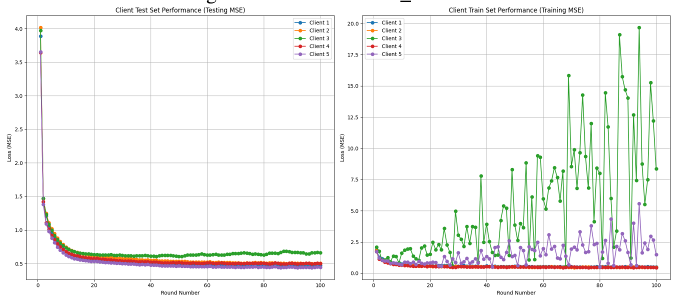
TABLE III
EVALUATION ON GD VS MINI-BATCH GD

GD vs mini(batch size)	Minimum MSE	#iterations when convergence
GD	0.506	20
mini 1500	0.504	30
mini 750	0.505	30
mini 300	0.502	50
mini 150	0.507	40
mini 1	0.584	above 100

Even though the loss is slightly less when we apply mini-batch and set the batch size to 300, 750 or 1500. But there were strange things happening on the training loss.

As shown in plot 3. A client's training loss experienced significant fluctuations. Generally, even without training, the loss would not exceed 5. However, as seen in the figure, the training loss astonishingly reached 20 (We assume that's because when mini-batch GD is set, the model becomes more unstable, as the data used by some clients may not be comprehensively learned). This situation also occurred at mini-batch sizes of 300, 750, and 1500. Therefore, we opted for using GD, which is more stable. The result of GD is shown in plot 4

Fig. 3. Plot: mini-batch_size=300



E. Evaluation on subsampling

We have fixed to use GD, set the learning rate to 0.05, set the number of epochs to 20, than evaluate on various subsampling values, from 0 to 4 ("no subsampling" to "randomly aggreagates models from 4 out of 5"). The result is shown in table IV

Fig. 4. Plot: Gradient Descent

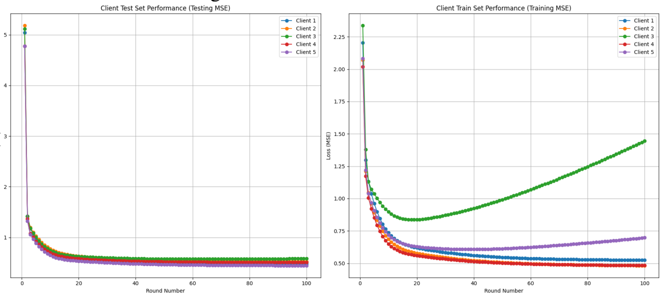


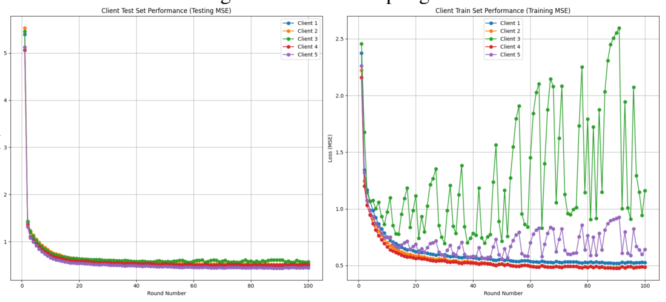
TABLE IV
EVALUATION ON SUBSAMPLING

subsampling value	Minimum MSE	#iterations when convergence
0	0.506	20
1	0.484	40
2	0.488	30
3	0.491	20
4	0.498	20

The table illustrates that activating subsampling leads to a decrease in loss as the subsampling value is reduced, however, this also results in an increase in the "number of iterations until convergence."

Here we are facing a trade-off between stability and the performance. The plot 4 shows the outcome when subsampling is deactivated, the performance is slightly worse compared to when subsampling is activated. However, the plot 5 shows the outcome of setting subsampling value to 4, even though the performance is slightly better, the training loss still experienced some fluctuations, but this time the values did not exceed 2.5 (If the subsampling value is reduced, the training loss will experience larger fluctuations, but they will not exceed 12 at most).

Fig. 5. Plot: subsampling=4



V. CONCLUSION

We have successfully implemented a federated learning system that allows clients to train local models and contribute to a global model, which achieved great performance on the test sets.

REFERENCES

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[2] Rohan Joseph. Grid search guideline. https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e, 2018.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] R. Kelley Pace and Ronald Barry. The clifornia housing dataset. https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html, 1997.