THE UNIVERSITY OF SYDNEY

Due: April 19th, 2024 (Friday, Week 8), by 11:59 PM

COMP3221
Assignment 2: Federated Learning

The main goal of this assignment is to implement a simple Federated Learning (FL) system. This project can be done in a group of two students, with only one team member submitting the work on behalf of the group. You need to register a group on the CANVAS *page:* ***COMP3221*** $\rightarrow$ ***People*** $\rightarrow$ ***Group - A2****.*

1 Learning Objectives

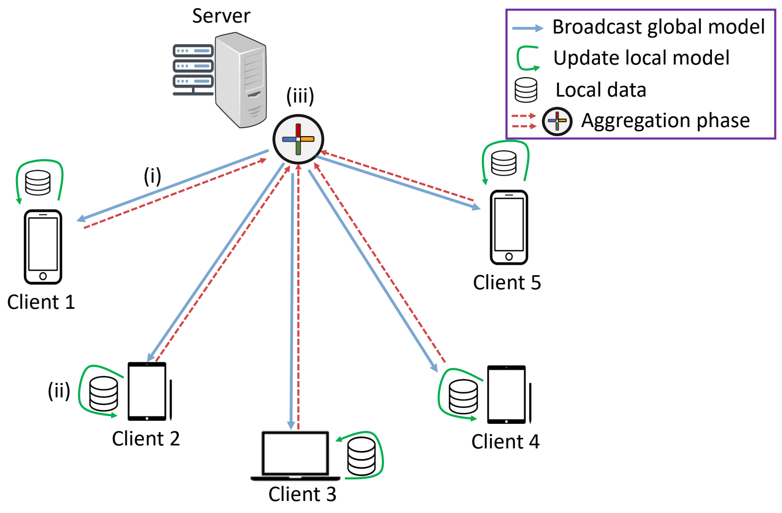


Figure 1: An example of a Federated Learning system with 1 server and 5 clients.

Your assignment involves developing a Federated Learning (FL) system that consists of one server and five clients as depicted in Fig. 1. 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.

On completing this assignment you will gain practical knowledge in:

- **Federated Learning principles**: Understand how to scale machine learning across multiple devices while ensuring data privacy and minimizing central data storage needs.
- **Client-Server programming**: Master the basics of network communications using sockets, including setting up connections, designing protocols, and handling network issues.

- **Machine learning programming**: Learn to implement, train, and evaluate machine learning models, focusing on practical aspects such as data handling, model and performance optimization.

2 Assignment Guidelines

2.1 Simulation Environment

Due to the unavailability of a physical network for deployment, you will simulate the FL on a single computer for both implementation and evaluation purposes. This simulation requires running separate instances of your program for each entity in the client-server architecture, using 'localhost' for communication. Specifically, each entity, including every client and the server, will be run in a different terminal window on your machine.

2.2 Federated Learning Algorithm

Algorithm 1 Federated Averaging (FedAvg)

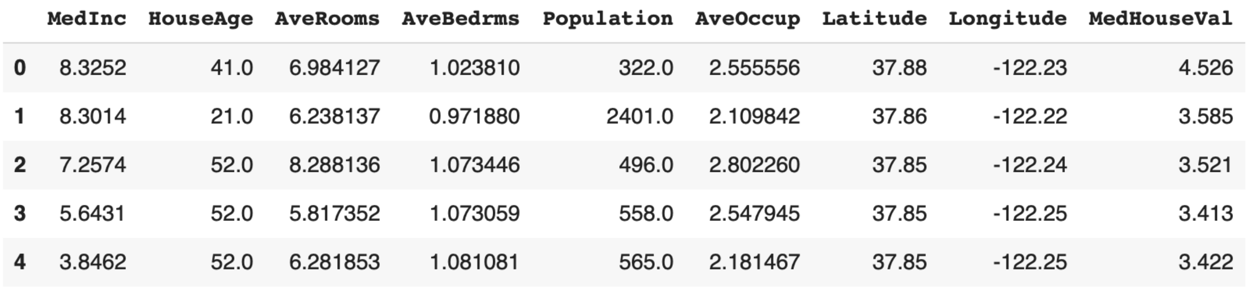
```
1: parameters: $K$ is number of clients, $E$ is number of local epoch, $n_k$ is local datasize of
   client $k$, $n$ is total datasize of $K$ clients.

2: procedure ServerUpdate                                                  ▷ Run on server
3:     Generate $w_0$ randomly
4:     for $t$ from $0$ to $T$ do
5:         Server broadcasts global model $w_t$ to $K$ clients
6:         for each client $k \in K$ in parallel do
7:             $w_{t+1}^k \leftarrow$ ClientUpdate($k$,$w_t$)
8:         end for
9:         Server receives new local models from $K$ clients and randomly selects
10:        a subset $M$ clients in $K$, $M \leq K$ to aggregates new global model:
11:        $w_{t+1} = \sum_{k=1}^{M} \frac{n_k}{n} w_{t+1}^k$
12:    end for
13: end procedure

14: procedure ClientUpdate($k$,$w_t$)                                      ▷ Run on client $k$
15:    for $e$ from $1$ to $E$ do
16:        Client $k$ updates local model $w_{t+1}^k$ based on the global model $w_t$
17:        using GD or Mini-Batch GD
18:    end for
19:    Client $k$ sends new local model $w_{t+1}^k$ to the server
20: end procedure
```

In this assignment, we will use the Federated Averaging (FedAvg) algorithm, a key approach in Federated Learning where client devices collaboratively train a model by computing updates locally and averaging these updates on a central server to improve the global model. The workings of FedAvg are elaborated in Algorithm 1. Here, K represents the total number of clients participating in the training process. T is the total number of global communication rounds between the clients and the server. w_t refers to the global model's parameters at iteration t, while w_{t+1}^k denotes the local model's parameters of client k at iteration $t+1$. E is the number of local epochs, i.e. the number of times each client goes through its entire dataset to train the model locally before sending updates to the global model. For local model training, clients can use either Gradient Descent (GD) or Mini-Batch GD as optimization methods.

2.3 Dataset and Model



	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Figure 2: Sammples of the California Housing Dataset.

For this assignment, we work with the California Housing Dataset[1], which is a widely recognized dataset used in machine learning for predicting house prices based on various features. This dataset contains 20640 data samples, which each include 8 features (median income, housing median age, average rooms, average bedrooms, population, average occupancy, latitude, and longitude) and 1 target variable (median house value) for different blocks in California. The dataset is insightful for understanding how house values vary by location and other factors.

To simulate an FL environment that reflects the heterogeneous nature of real-world data, we have distributed the dataset across $K = 5$ clients. Each client receives a portion of the dataset, varying in size, to mimic the diversity in data distribution one might encounter in practical FL scenarios. The federated dataset is prepared and accessible in `FLData.zip`, available for download on the page CANVAS $\rightarrow$ Assignment 2. For every client, we provide two CSV files: one for training set and one for testing set. For instance, the training and testing data for Client 1 are named "`calhousing_train_client1.csv`" and "`calhousing_test_client1.csv`", respectively.

Considering the objective is a regression problem focused on predicting house values, a Linear Regression model is apt for this task. It efficiently models the correlation between house features and their prices. The ultimate goal is to train a Linear Regression model optimized across the distributed datasets.

[1]https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html

2.4 Program Structure

This assignment involves developing two main programs: one for the server and another for the clients. It's essential to start the server program before running any client programs to ensure proper communication and data exchange.

2.4.1 Server

The server program, named **`COMP3221_FLServer.py`** requires two command-line arguments for execution as follows.

```
python COMP3221_FLServer.py <Port-Server> <Sub-Client>
```

- **<Port-Server>**: The port number on which the server listens for incoming model updates from the clients. For this assignment, it is set to 6000.
- **<Sub-Client>**: An integer value determines whether client subsampling is enabled. A value of 0 means no subsampling, so the server aggregates models from all clients. A value of M $(0 < M < K)$ activates subsampling, where the server randomly aggregates models from only M out of the K clients.

Example usage:

```
python COMP3221_FLServer.py 6000 2
```

2.4.2 Client

The client program, named **`COMP3221_FLClient.py`**, accepts the following command line arguments:

```
python COMP3221_FLClient.py <Client-id> <Port-Client> <Opt-Method>
```

- **Client-id**: The identifier for a client in a FL network which is indexed sequentially as client1, client2, client3, client4, and client5.
- **Port-Client**: The port number used by the client to receive model updates from the server. Port numbers are assigned starting at 6001 for client1 and increment by one for each subsequent client, up to 6005 for client5.
- **Opt-Method**: The optimization method used for local model training. A value of 0 selects Gradient Descent (GD), and a value of 1 selects Mini-Batch GD.

Example Usage:

```
python COMP3221_FLClient.py client1 6001 1
```

2.5 Assignment Tasks

2.5.1 Server

Following the FedAvg algorithm (Alg. 1), at the beginning, the server initially generates a global Linear Regression model with random parameters, denoted as w_0. It then starts listening for initial connection requests ("hand-shaking messages") from clients wanting to join the Federated Learning (FL) system. These messages should include information about their data size and ID, giving the server insight into the participating clients.

Once the server receives a handshake message from one client, it will continue to wait for 30 seconds to allow more client registrations. This waiting period occurs only once at the server's startup, ensuring a sufficient number of clients involved in the training process. Following this, the server broadcasts the global model to all registered clients and waits for the return of their new local models for aggregation.

After receiving local models from every registered client, the server aggregates these models to update a new global model. Depending on the specific configuration, this aggregation may involve models from all clients or just a selected subset of $M < K$ clients. Once updated, the server broadcasts this new global model to all registered clients, marking the completion of one global communication round.

In this assignment, the FL system will run for T global communication rounds. Upon completing these rounds, the server will broadcast a "finish message" to all clients, signaling them to stop the training process.

For each global round, the server will print out the following output to the terminal:

```
Global Iteration 10:
Total Number of clients: 5
Getting local model from client 1
Getting local model from client 2
Getting local model from client 5
Getting local model from client 4
Getting local model from client 3
Aggregating new global model
Broadcasting new global model
```

The server is responsible for managing clients and should keep a list that contains information about registered clients. If a new client attempts to register after the server has completed its initialization phase, the server will add this client's information to the current client list and share the global model with them in the next global communication round.

2.5.2 Client

Upon starting up, each client loads its own dataset and registers with the server by sending a hand-shaking message. Once the global model is received, the client first evaluates this model using its local test data. After that, it utilizes this global model as an initial point to train an updated local model. The local training process can be finished in E local epochs using optimization methods such as GD or Mini-Batch GD. Subsequently, the client sends this newly trained local model to the server and waits to receive the next iteration of the global model.

During each global communication round, the client outputs the following to the terminal:

```
I am client 1
Received new global model
Testing MSE: 0.0052
Local training...
Training MSE: 0.0012
Sending new local model
```

Additionally, it logs the training and testing Mean Square Error (MSE) results for each round in a file named **`<Client-id>_log.txt`**, serving as a means for later evaluation.

Important Notes:

- You have the flexibility to define the format of hand-shaking messages and data packets used for model exchange between the server and the clients.
- You are allowed to use Machine Learning libraries (e.g., Scikit-learn, Pytorch) that were introduced during the tutorials for your implementation.
- You have the flexibility to select the values for input parameters, including the number of training rounds (T), the number of epochs (E), the learning rate, and the batch size. By adjusting these parameters, you can optimize the training process to ensure that your model achieves high performance.

3 Report and Submission

3.1 Report

Your submission must include a report document that concisely describes your work within a strict limit of no more than 3 pages, with the exception of the references section, which may extend beyond this page limit.

Your report should present your understanding of the algorithm, model, and dataset, alongside your approach to implementation. It should also include an insightful discussion on the experiment outcomes and a comparative analysis evaluating the performance of the global model across various scenarios.

Here are some example scenarios you can explore in your experiments:

1. Evaluate the performance of the global Linear Regression model across each client's test dataset.
2. Examine the differences in utilizing Gradient Descent (GD) versus Mini-Batch GD; considering various batch sizes and learning rate.
3. Analyze the impact of subsampling a subset of clients ($M < K$) compared to involving all clients ($M = K$) in the training process.

We recommend using figures and tables to visualize the experimental results. For example, you can demonstrate the convergence of training and testing MSE over iterations to provide a clear representation of the model's performance improvement over time.

3.2 Submission Files

You are required to submit your source code and a short report to CANVAS.

- Code (a zipped archive contains all your code files, **no need to submit the data files**) **`SSID_COMP3221_Code.zip`**.
- Code Text (a single *.txt* file includes all implementation code for Plagiarism checking) **`SSID_COMP3221_Code.txt`**.
- Readme (A detailed *.txt* file that outlines the coding environment, version of packages used, instructions to run your program, and commands to reproduce the experimental results.) **`SSID_COMP3221_Readme.txt`**.
- Report (A *.pdf* file that includes all content required in the report section) **`SSID_COMP3221_Report.pdf`**.

Note that you must upload your submission BEFORE the deadline. The CANVAS would continue accepting submissions after the due date; however, late submissions would incur a penalty per day with a maximum of 5 days late submission allowed.

4 Academic Honesty / Plagiarism

By uploading your submission to CANVAS you implicitly agree to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not your work you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

In assessing a piece of submitted work, the School of Computer Science may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

5 Marking

This assignment contributes 15% to your final grade for this unit of study. The distribution of marks between the assignment components is as follows.

- Code: 70%.
- Report: 30%.

Please refer to the *rubric* in Canvas (**COMP3221** → **Assignment** → **Assignment 2** → **Assignment 2 - Rubric**) for detailed marking scheme.

6 Feedback

After Assignment 2 marks come out, please submit your inquiries about marking within one week. All inquiries after that will NOT be responded.