Data Science Hw4

111061702 常安彥

1. Code
   a. Severbase.py - select_users

```
def select_users(self, round, num_users):
    if num_users > len(self.users):
        num_users = len(self.users)
    #####                        Random Sample                        #####
    # return random.sample(self.users, num_users)
    #####                                                             #####

    #####                        Round Robin                          #####
    count = self.temp
    diff = len(self.users) - count
    if diff > num_users:
        self.temp += num_users
        return self.users[count: self.temp]
    else:
        self.temp = num_users - diff
        return self.users[count:] + self.users[:self.temp]
    #####                                                             #####
```

First, make sure that the number of selected users isn't more than the total users. Next, I implemented random sampling with the 'sample' function in the 'random' package.

Finally, I implemented 'Round Robin.' I set a parameter, temp, in the initialization of the server, and this represents the first selected user in current round. Then during the round, the selected user starts from 'temp,' and records the following ones. Update the 'temp' user at last.

   b. Severbase.py – aggregate_parameters

```
def aggregate_parameters(self):
    sum_sample = sum(user.train_samples for user in self.selected_users)
    aggregate_params = {}
    for user in self.selected_users:
        weight = user.train_samples / sum_sample
        user_params = user.model.state_dict()
        # print(user_params)
        for k in user_params.keys():
            if k not in aggregate_params.keys():
                aggregate_params[k] = weight * user_params[k]
            else:
                aggregate_params[k] += weight * user_params[k]
    # print(aggregate_params)
    self.model.load_state_dict(aggregate_params)
```

First, calculate each weight of the users according to their training samples. Then, I create a new dictionary to store the parameters of the models from the selected users corresponding to their weights. Finally, load the state dictionary to update the model of the server.

c. Userbase.py – set_parameters

```
def set_parameters(self, model, beta=1):
    server_params = model.state_dict()
    user_params = self.model.state_dict()
    new_params = {}
    for k in user_params.keys():
        new_params[k] = beta * server_params[k] + (1 - beta) * user_params[k]
    self.model.load_state_dict(new_params)
```

I create a new dictionary, and add up the parameters of the server model and the user model in proportional of beta. Later on, the model of the user loads the dictionary as the new parameters.

2. Data Distribution

Below is the result after running Dirichlet function with alpha equals to 0.1 and 50. It is obvious that the distribution of the first case(0.1) is very imbalanced, where most of them are smaller than 0.1 and one of them is 0.488. The second case(50) is very balanced, where all of them are near 0.1, showing the class is equally shared to all users.

```
alpha = 0.1:  [4.32688012e-02 1.52119315e-02 6.71905743e-01 3.73881593e-02
 4.53979936e-04 1.45191764e-06 2.27280075e-01 1.06712618e-08
 3.66532278e-03 8.24524485e-04]
alpha = 50:  [0.12105555 0.09105766 0.10224088 0.08541914 0.10144713 0.10334578
 0.10078    0.09940104 0.10115487 0.09409797]
```

The weights of the parameters updated from the participants are determined by their data volume, so participants with more samples will have a larger impact on the global model. If samples of a certain class are rare among most users, the global model may not fully capture the features and patterns of the class.

Furthermore, if the samples of a certain class are distributed unevenly among the users, the predictions may be affected by the bias. When only few users provide samples of the specific class during training process, the global model may perform poorly on it due to lack of sufficient training opportunities to learn the patterns of the class.

a. alpha = 0.1 Best accuracy: 0.3569

```
————————Round number: 149 ————————


Average Global Accuracy = 0.2611, Loss = 1.85.
Best Global Accuracy = 0.3569, Loss = 1.81, Iter = 145.
Finished training.
```

b. alpha = 50 Best accuracy: 0.8040

```
————————Round number: 149 ————————


Average Global Accuracy = 0.7999, Loss = 0.74.
Best Global Accuracy = 0.8040, Loss = 0.71, Iter = 140.
Finished training.
```

3. Number of Users per Round
    a. Accuracy

        Aggregating parameters from ten models at once allows more samples data to be used for training and updating the models. With each model receiving a richer and more diverse set of data, there is a potential for learning more comprehensive features and patterns. This helps improve the robustness of the model, enabling it to better handling different data distribution and noise.

        Num users = 2:

```
————————Round number: 149 ————————


Average Global Accuracy = 0.6527, Loss = 1.03.
Best Global Accuracy = 0.7066, Loss = 0.86, Iter = 145.
Finished training.
```

        Num users = 10:

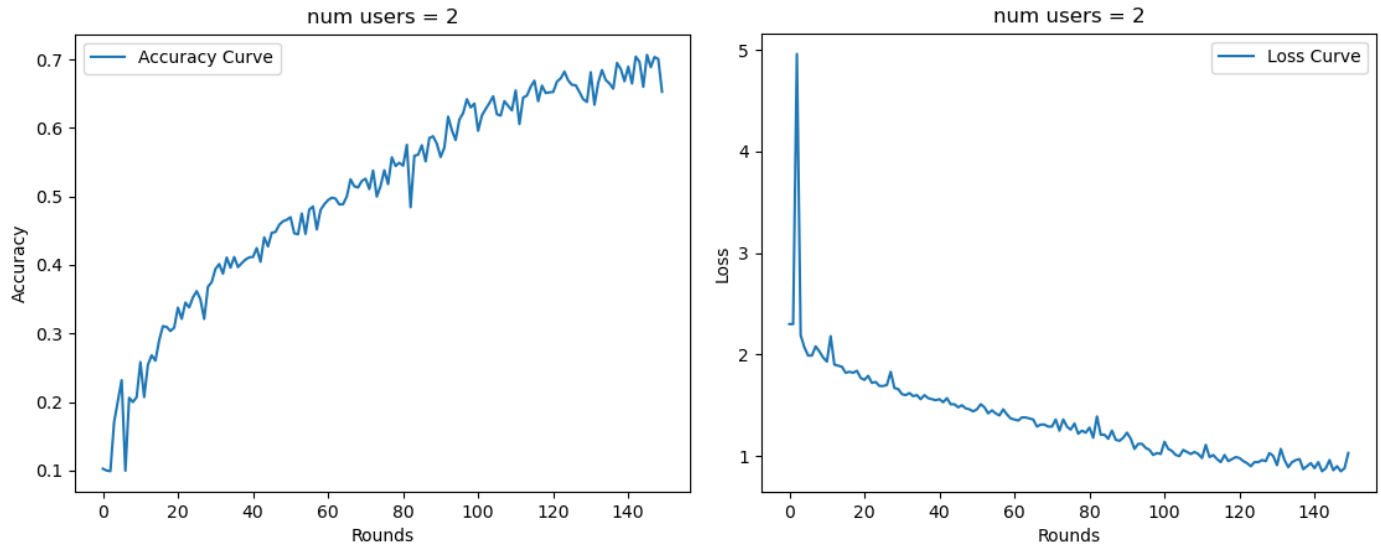
```
————————Round number: 149 ————————


Average Global Accuracy = 0.8020, Loss = 0.76.
Best Global Accuracy = 0.8088, Loss = 0.68, Iter = 129.
Finished training.
```

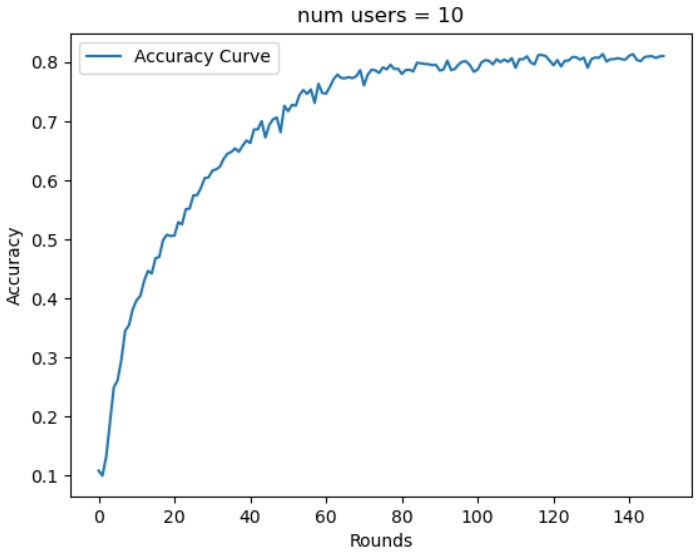
    b. Convergence

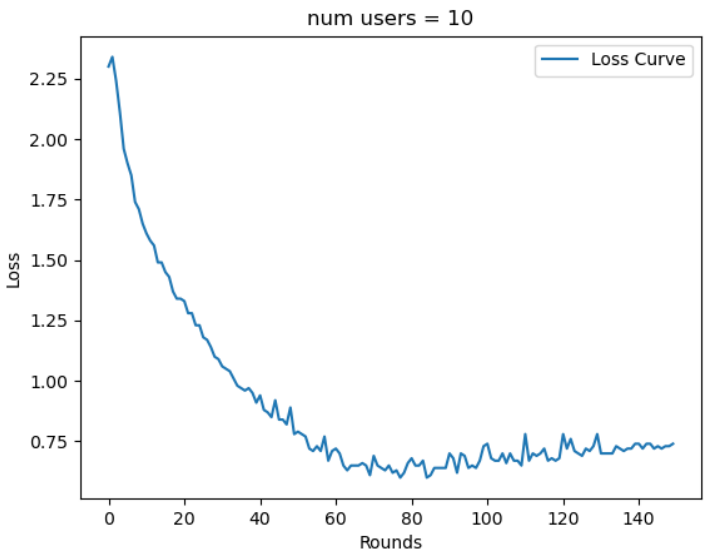
        When selecting more models for parameter aggregation, more sample data is used to update the model parameters, leading to potentially more stable and accurate average updates. This can facilitate faster convergence to better performance level and improve the converging speed.

        Num users = 2 accuracy and loss curve:



Num users = 10 accuracy and loss curve:





4. Accuracy

   Average: 0.8057

   Best: 0.8171

```
————————Round number: 149 ————————


Average Global Accuracy = 0.8057, Loss = 0.76.
Best Global Accuracy = 0.8171, Loss = 0.68, Iter = 125.
Finished training.
```

5. What I Have Learnt

   a. Horizontal Federated Learning

      I have learnt about how to implement horizontal federated learning. When tracing the code, I more clearly understood the reaction between server and users. After coding, I completely understood how parameters aggregate when updating global models.

   b. Distribute Data with Dirichlet

      I have learnt about the method of distributing data for each class. Assigning the alpha was the important part in order to equally sample data.

   c. Relationship between Performance and Number of Users, Data Distribution

      The data distribution may affect the robustness of the model due to lack of some features and patterns during training. Also, the number of users affect the speed of convergence.