In [36]:

```
# Import the required libraries:
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, random_split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np
```

In [37]:

```
# Preprocess the dataset:
class PaddyDataset(Dataset):
    def __init__(self, dataframe):
        self.data = dataframe
        self.X = self.data.iloc[:, :-1].values
        self.y = self.data.iloc[:, -1].values

        # Standardize features by removing the mean and scaling to unit variance
        scaler = StandardScaler()
        self.X = scaler.fit_transform(self.X)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return torch.tensor(self.X[idx], dtype=torch.float32), torch.tensor(self.y[idx]
```

In [38]:

```
# Load Paddy Field dataset
dataframe = pd.read_csv(r"C:\Users\saldr\Downloads\paddyfield.csv", header=None)
#dataframe.columns = ["Water Level", "Moisture Percentage", "Light Intensity", "Tempera
```

In [39]:

```
# Split dataset into training and test sets
train_df, test_df = train_test_split(dataframe, test_size=0.35, random_state=42)

# Create dataset objects
train_dataset = PaddyDataset(train_df)
test_dataset = PaddyDataset(test_df)
len(train_dataset)
```

Out[39]:

```
6840
```

In [40]:

```
# Split training dataset among clients
num_clients = 5
client_datasets = torch.utils.data.random_split(train_dataset, [len(train_dataset) // n
client_datasets
```

Out[40]:

```
[<torch.utils.data.dataset.Subset at 0x18f29f56160>,
 <torch.utils.data.dataset.Subset at 0x18f29f56220>,
 <torch.utils.data.dataset.Subset at 0x18f29f566a0>,
 <torch.utils.data.dataset.Subset at 0x18f29f561c0>,
 <torch.utils.data.dataset.Subset at 0x18f29f565b0>]
```

In [58]:

```
# Define the Model
# Define a simple neural network for the classification task:
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        #self.fc1 = nn.Linear(8, 64)
        #self.fc2 = nn.Linear(64, 32)
        #self.fc3 = nn.Linear(32, 2)
        self.fc1 = nn.Linear(5, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 32)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [64]:

```
# Client Update Function
# Define a function to train the model on each client:
def client_update(client_model, optimizer, train_loader, epochs=1):
    client_model.train()
    criterion = nn.CrossEntropyLoss()
    for epoch in range(epochs):
        for data, target in train_loader:
           # print(target.shape)
            optimizer.zero_grad()
            output = client_model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
```

In [65]:

```
# Server Aggregation Function
# Define a function to aggregate the models from each client:
def server_aggregate(global_model, client_models):
    global_dict = global_model.state_dict()
    for k in global_dict.keys():
        global_dict[k] = torch.stack([client_models[i].state_dict()[k].float() for i in
    global_model.load_state_dict(global_dict)
```

In [66]:

```
# Evaluation Function
# Define a function to evaluate the global model on the test dataset:
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
    return correct / total
```

In [67]:

```
# Federated Learning Process
# Implement the federated learning process with accuracy evaluation:
def federated_learning(global_model, client_datasets, num_rounds=10, epochs_per_client=
    client_models = [SimpleNN() for _ in range(num_clients)]
    client_optimizers = [optim.SGD(model.parameters(), lr=0.01) for model in client_mod

    test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

    for round in range(num_rounds):
        print(f"Round {round+1}")
        # Train each client model locally
        for i in range(num_clients):
            train_loader = DataLoader(client_datasets[i], batch_size=32, shuffle=True)
            client_update(client_models[i], client_optimizers[i], train_loader, epochs=

        # Aggregate the client models into the global model
        server_aggregate(global_model, client_models)

        # Evaluate the global model
        accuracy = evaluate_model(global_model, test_loader)
        print(f"Accuracy after round {round+1}: {accuracy:.4f}")

        # Update client models with the aggregated global model
        for model in client_models:
            model.load_state_dict(global_model.state_dict())
```

In [68]:

```
# Initialize and Start Training
# Initialize the global model and start the federated learning process:
global_model = SimpleNN()
client_datasets
federated_learning(global_model, client_datasets, num_rounds=10, epochs_per_client=1)
```

```
Round 1
Accuracy after round 1: 0.4389
Round 2
Accuracy after round 2: 0.4389
Round 3
Accuracy after round 3: 0.4389
Round 4
Accuracy after round 4: 0.4389
Round 5
Accuracy after round 5: 0.4389
Round 6
Accuracy after round 6: 0.4389
Round 7
Accuracy after round 7: 0.4389
Round 8
Accuracy after round 8: 0.4389
Round 9
Accuracy after round 9: 0.4389
Round 10
Accuracy after round 10: 0.6251
```

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```