Course: IN5460/IN9460
Artificial Intelligence for Energy Informatics

Assignment 2

Appliance energy consumption prediction and classification using federated learning

Delivery format:
(i) Github link (preferred) or a zip file of your developed code.
(ii) A PDF report of your solution and result description.

Delivery and evaluation info:
Work in assigned groups of 3-4 students.
Students in the same group receive the same score.
Assignment 2 counts for 20% of the grade.

Deliver to:
Sabita Maharjan, `sabita@ifi.uio.no`
Shiliang Zhang[1], `shilianz@ifi.uio.no`

Delivery deadline: **12.11.2023 kl 23:59**[2]

October 2023

[1]Please contact Shiliang Zhang (shilianz@ifi.uio.no) if you have questions regarding this assignment.
[2]Please notice that this is a firm deadline. Individuals/Groups that send reports after the deadline will receive the grade "F".

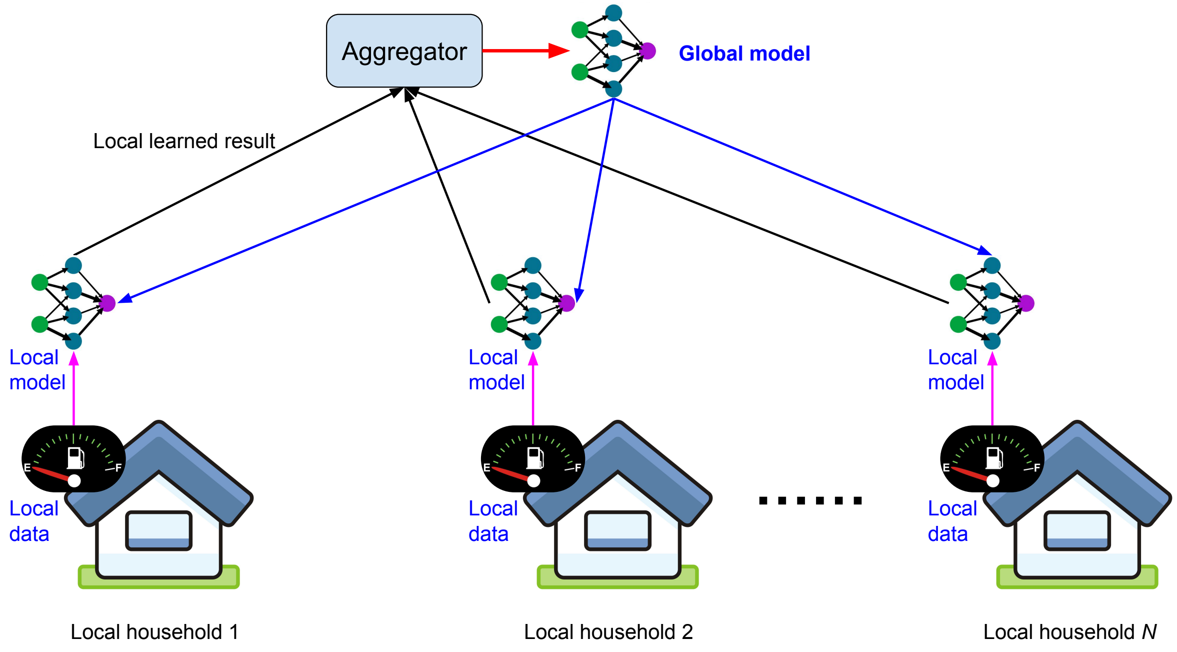

Figure 1: System architecture

1 About this assignment

In this assignment, you will use federated learning (FL) to (i) train a model that can predict the energy consumption of appliances in a household, and (ii) train another model that can classify the type of electricity appliances according to their energy consumption recordings.

Scenario: as shown in the figure 1, we have several households, all of which have common appliances and the energy consumption of those appliances are recorded as time series. We want to (i) train a prediction model that takes input as time-series of energy consumption by appliances in a household during seven consecutive days (from 00:00:00 of the first day to 23:59:59 of the last day), and outputs the energy consumption of appliances for the 8-th day (from 00:00:00 to 23:59:59) and (ii) train a classification model whose input is the energy consumption of a certain appliance during one day (from 00:00:00 to 23:59:59), and the output is the label of the appliance type.

The goal is to train global models from distributed data silos-or households in this assignment-while avoiding consumption data transferring. Under the FL framework, each house will train a local model based on their own data, and then we aggregate the local models to be a global one.

The following sections provide instructions and tools that might help. Note that such information is associated with the usage of Python. But you can use your preferred programming language when developing your own solution.

2 General information, instructions, and requirements

To translate the assignment into a FL task, there is a need to formulate the described scenario and interactions. We provide the following data, pseudo code in section 3.1, and example code snippets in section 3.4 that might help.

2.1 Dataset

The dataset to be used in this assignmen is derived from Dataset of an Energy Community's Consumption and Generation with Appliance Allocation. You can download the dataset HERE.

This dataset records appliance energy consumption in one year from 50 households. The consumption is recorded every 15 minutes. Each household has the following 10 appliance records: air conditioner (AC), dish washer, washing machine, dryer, water heater, TV, microwave, kettle, lighting, and refrigerator.

2.2 Data pre-processing

To train and test the federated learning model, you need to pre-process the data to get the input and output data so that you can feed the learning procedures.

For the prediction model

You need to sum up the energy consumption from all appliances to obtain the appliance consumption for a certain household over the time. Then you need to segment the summed data to gain the input and output time series to feed the learning, e.g., one input as the data from Monday to Sunday, and the corresponding output as the data for the next Monday. You move the sampling window one day forward and you then obtain another input as data from Tuesday to the next Monday, and the corresponding output as the data for the next Tuesday, and so son.

For the classification model

You need to partition the one-year energy consumption into time series of daily consumption for each appliance. For a single time series of appliance daily consumption, you need to assign a label so as to distinguish which appliance is the time series generated from.

For the data you prepared for both the prediction and classification model, you need to randomly select 80% of the household appliance consumption data as training data (in the case of classification, 80% of daily consumption data for each appliance in each household), and the remaining 20% as test data. You have the flexibility whether to use validation data, which can be obtained by further partitioning the training data.

2.3 Contents of the report to be delivered

The delivered report should include the following.

Federated learning model establishment, training, and result visualization

You need to use both LSTM and another RNN (e.g., a vanilla RNN) when implementing federated learning for both the prediction and classification model. You will determine how many layers and how many nodes in each layer when designing the model structure. The following questions should be considered in your assignment report.

Question 1.1: prediction performance of federated learning based on RNN

Is the federated learning efficient in this scenario of appliance energy consumption prediction? Please use simulation plots to show the how the training error varies during the training process using training (or training and validation) data. You can use mean square error (MSE) or other specified measurement for the calculation of error. Please use simulation plots to show the predicted output vs. ground truth during the model training and testing. Please discuss whether the performance of model training can be improved by adding more epochs or through other configuration changes.

Question 1.2: prediction performance of federated learning based on LSTM

Keep the same settings as in Question 1.1 except that you use LSTM rather than RNN. Please use simulation plots to show how the training error varies over time, and use simulation plots to show the predicted output vs. ground truth during the model training and testing. Compare the performance with that of RNN regarding execution time and prediction error during the test.

Question 2.1: classification accuracy of federated learning based on RNN

Is the federated learning efficient in this scenario of appliance classification? Please use simulation plots to show the classification accuracy during the training process using training (or training and validation) data, and show the classification accuracy of the trained model using test data. Please discuss whether the accuracy during the training and testing can be improved by adding more epochs or through other configuration changes.

Question 2.2: which appliance is better classified in Question 2.1 and which is not?

The accuracy in Question 2.1 manifests how the model works for all the appliance as a whole. You also need to show how the classification works for each appliance. To do so, you need to generate the confusion matrix of the classification result, both for the model training and testing. The confusion matrix should present the classification accuracy for each appliance.

Question 2.3: how the results vary when using LSTM?

Keep the same settings as in Question 2.1 and 2.2, except that you use LSTM when implementing federated learning. You then show the simulation plots as required in Question 2.1 and 2.2, and compare with the results when using RNN.

3 Pseudo code, libraries, and code snippets

3.1 Pseudo code for the implementation

Algorithm 1 Pseudo code: household appliance consumption prediction using FL and RNN

```
Require:
  Initialize the structure of RNN
  Initialize the global model $W_0$
  for $epoch = 1, 2, \ldots$ do
      for $batch = 1, 2, \ldots$ do
          for $household = 1, 2, \ldots, 50$ do
              Calculate the gradient for the local model of the household
          end for
          Aggregate the gradients from all households by averaging them
          Update the global model using the aggregated gradients
          Send the updated global model to each household as their local model
      end for
  end for
```

3.2 Libraries (Python)

- Numpy: Enable array computation and diverse mathematical functions. An explanation of array can be found here.
- Tensorflow: Enable machine/deep learning from loading dataset, building models, to model training and logging.
- Sktime: Enable time-series analysis, e.g., classification, regression, clustering.
- pytablereader: a Python library to load structured table data from files/strings/URL with various data format: CSV / Excel / Google-Sheets / HTML / JSON / LDJSON / LTSV / Markdown / SQLite / TSV.

3.3 Federated learning toolkits

We provide five Python toolkits for FL, which might help your solution development.

- Flower: a friendly federated learning framework that can be used across different frameworks, e.g., PyTorch, TensorFlow, scikit-learn, Pandas, NumPy, etc.
- FederatedScope: a comprehensive federated learning platform that provides convenient usage and flexible customization for various federated learning tasks.
- OpenFL: a Python 3 framework for Federated Learning designed to be a flexible, extensible and easily learnable tool for data scientists.
- TensorFlow Federated: enables developers to use the included federated learning algorithms with their models and data.
- FedML: provides a research and production integrated edge-cloud platform for Federated/Distributed Machine Learning different scales.

3.4 Code snippets (Python)

Below are several code snippets that might provide clues for your coding.

3.4.1 Pre-process the dataset

Listing 1: Python example of data pre-processing. This code snippet shows how to load dataset and formulate the data to fit the meachine learning procedures.

```python
import pytablereader as ptr
import pytablewriter as ptw

file_path = "sample_data.csv"

with open(file_path, "w") as f:
    f.write(csv_text)

loader = ptr.CsvTableFileLoader(file_path)
for table_data in loader.load():
    print("\n".join([
        "load from file",
        "==============",
        "{:s}".format(ptw.dumps_tabledata(table_data)),
    ]))
```

3.4.2 Define model structure

Listing 2: Python example of constructing an RNN model for classification purpose. This code shows how to build an RNN model and set layers for this model using the tensorflow library.

```python
import tensorflow as tf
from tensorflow.contrib import rnn
from tensorflow.contrib import legacy_seq2seq
```

```
import numpy as np


class Model():
    def __init__(self, args, training=True):
        self.args = args
        if not training:
            args.batch_size = 1
            args.seq_length = 1

        # choose different rnn cell
        if args.model == 'rnn':
            cell_fn = rnn.RNNCell
        elif args.model == 'gru':
            cell_fn = rnn.GRUCell
        elif args.model == 'lstm':
            cell_fn = rnn.LSTMCell
        elif args.model == 'nas':
            cell_fn = rnn.NASCell
        else:
            raise Exception("model type not supported: {}".format(args.
model))

        # warp multi layered rnn cell into one cell with dropout
        cells = []
        for _ in range(args.num_layers):
            cell = cell_fn(args.rnn_size)
            if training and (args.output_keep_prob < 1.0 or args.
input_keep_prob < 1.0):
                cell = rnn.DropoutWrapper(cell,
                                          input_keep_prob=args.
input_keep_prob,
                                          output_keep_prob=args.
output_keep_prob)
            cells.append(cell)
        self.cell = cell = rnn.MultiRNNCell(cells, state_is_tuple=True)

        # input/target data (int32 since input is char-level)
        self.input_data = tf.placeholder(
            tf.int32, [args.batch_size, args.seq_length])
        self.targets = tf.placeholder(
            tf.int32, [args.batch_size, args.seq_length])
        self.initial_state = cell.zero_state(args.batch_size, tf.float32)

        # softmax output layer, use softmax to classify
        with tf.variable_scope('rnnlm'):
            softmax_w = tf.get_variable("softmax_w",
                                        [args.rnn_size, args.vocab_size])
            softmax_b = tf.get_variable("softmax_b", [args.vocab_size])
        # transform input to embedding
        embedding = tf.get_variable("embedding", [args.vocab_size, args.
    rnn_size])
        inputs = tf.nn.embedding_lookup(embedding, self.input_data)

        # dropout beta testing: double check which one should affect next
    line
        if training and args.output_keep_prob:
            inputs = tf.nn.dropout(inputs, args.output_keep_prob)

        # unstack the input to fits in rnn model
        inputs = tf.split(inputs, args.seq_length, 1)
        inputs = [tf.squeeze(input_, [1]) for input_ in inputs]

        # loop function for rnn_decoder, which take the previous i-th cell
    's output and generate the (i+1)-th cell's input
        def loop(prev, _):
            prev = tf.matmul(prev, softmax_w) + softmax_b
            prev_symbol = tf.stop_gradient(tf.argmax(prev, 1))
            return tf.nn.embedding_lookup(embedding, prev_symbol)

        # rnn_decoder to generate the ouputs and final state. When we are
    not training the model, we use the loop function.
        outputs, last_state = legacy_seq2seq.rnn_decoder(inputs, self.
    initial_state, cell, loop_function=loop if not training else None,
    scope='rnnlm')
        output = tf.reshape(tf.concat(outputs, 1), [-1, args.rnn_size])

        # output layer
        self.logits = tf.matmul(output, softmax_w) + softmax_b
        self.probs = tf.nn.softmax(self.logits)

        # loss is calculate by the log loss and taking the average.
        loss = legacy_seq2seq.sequence_loss_by_example(
                [self.logits],
                [tf.reshape(self.targets, [-1])],
                [tf.ones([args.batch_size * args.seq_length])])
        with tf.name_scope('cost'):
            self.cost = tf.reduce_sum(loss) / args.batch_size / args.
    seq_length
        self.final_state = last_state
        self.lr = tf.Variable(0.0, trainable=False)
        tvars = tf.trainable_variables()

        # calculate gradients
        grads, _ = tf.clip_by_global_norm(tf.gradients(self.cost, tvars),
                args.grad_clip)
        with tf.name_scope('optimizer'):
            optimizer = tf.train.AdamOptimizer(self.lr)

        # apply gradient change to the all the trainable variable.
        self.train_op = optimizer.apply_gradients(zip(grads, tvars))

        # instrument tensorboard
        tf.summary.histogram('logits', self.logits)
        tf.summary.histogram('loss', loss)
        tf.summary.scalar('train_loss', self.cost)
```

Listing 3: Python example of constructing an LSTM model for classification purpose. This code shows how to build an LSTM model and set layers for this model using the tensorflow library.

```
import tensorflow as tf
class LSTM(tf.keras.Sequential):
    def __init__(self):
        super(LSTM, self).__init__()

    def build_default(self, input_shape, n_classes,
    use_batch_shape=False, activation='softmax', verbose=0):
        model = tf.keras.models.Sequential()
        if use_batch_shape:
            model.add(tf.keras.layers.Bidirectional
            (tf.keras.layers.LSTM(512, return_sequences=True),
            batch_input_shape=input_shape))
        else:
            model.add(tf.keras.layers.Bidirectional
            (tf.keras.layers.LSTM(512, return_sequences=True),
            input_shape=input_shape))

        model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(512))
   )
        model.add(tf.keras.layers.Dense(n_classes, activation=activation))

        if verbose:
            model.summary()
        return model
```

Listing 4: Python example of constructing federated learning architecture.

```
import tensorflow as tf
import tensorflow_federated as tff

# Load simulation data.
source, _ = tff.simulation.datasets.emnist.load_data()
def client_data(n):
  return source.create_tf_dataset_for_client(source.client_ids[n]).map(
      lambda e: (tf.reshape(e['pixels'], [-1]), e['label'])
  ).repeat(10).batch(20)

# Pick a subset of client devices to participate in training.
train_data = [client_data(n) for n in range(3)]

# Wrap a Keras model for use with TFF.
```

```python
def model_fn():
  model = tf.keras.models.Sequential([
      tf.keras.layers.Dense(10, tf.nn.softmax, input_shape=(784,),
                            kernel_initializer='zeros')
  ])
  return tff.learning.from_keras_model(
      model,
      input_spec=train_data[0].element_spec,
      loss=tf.keras.losses.SparseCategoricalCrossentropy(),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

# Simulate a few rounds of training with the selected client devices.
trainer = tff.learning.build_federated_averaging_process(
  model_fn,
  client_optimizer_fn=lambda: tf.keras.optimizers.SGD(0.1))
state = trainer.initialize()
for _ in range(5):
  state, metrics = trainer.next(state, train_data)
  print(metrics['train']['loss'])
```