federated_learning_evaluation_methodology

June 18, 2020

1 Evaluation Methodology of Federated Learning

In this notebook we study the different evaluation methodologies that we can use when we want to evaluate FL problems. First, we set up the FL configuration (for more information see Basic Concepts Notebook).

[1]:
```
import shfl
import tensorflow as tf
import numpy as np
import random

random.seed(123)
np.random.seed(seed=123)

class Reshape(shfl.private.FederatedTransformation):

    def apply(self, labeled_data):
        labeled_data.data = np.reshape(labeled_data.data, (labeled_data.data.
 ↪shape[0], labeled_data.data.shape[1], labeled_data.data.shape[2],1))

class Normalize(shfl.private.FederatedTransformation):

    def __init__(self, mean, std):
        self.__mean = mean
        self.__std = std

    def apply(self, labeled_data):
        labeled_data.data = (labeled_data.data - self.__mean)/self.__std

def model_builder():
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',␣
 ↪activation='relu', strides=1, input_shape=(28, 28, 1)))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2,␣
 ↪padding='valid'))
    model.add(tf.keras.layers.Dropout(0.4))
    model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',␣
 ↪activation='relu', strides=1))
```

```
    model.add(tf.keras.layers.MaxPooling2D(pool_size=2, strides=2,␣
↪padding='valid'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.1))
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    model.compile(optimizer="rmsprop", loss="categorical_crossentropy",␣
↪metrics=["accuracy"])

    return shfl.model.DeepLearningModel(model)
```

[2]:
```
#Read data
database = shfl.data_base.Emnist()
train_data, train_labels, test_data, test_labels = database.load_data()

#Distribute among clients
non_iid_distribution = shfl.data_distribution.NonIidDataDistribution(database)
federated_data, test_data, test_labels = non_iid_distribution.
↪get_federated_data(num_nodes=5, percent=10)

#Set up aggregation operator
aggregator = shfl.federated_aggregator.FedAvgAggregator()
federated_government = shfl.federated_government.
↪FederatedGovernment(model_builder, federated_data, aggregator)

#Reshape and normalize
shfl.private.federated_operation.apply_federated_transformation(federated_data,␣
↪Reshape())

mean = np.mean(test_data.data)
std = np.std(test_data.data)
shfl.private.federated_operation.apply_federated_transformation(federated_data,␣
↪Normalize(mean, std))
```

1.1 Evaluation methodology 1: Global test dataset

The first evaluation methodology that we propose consists of the federated version of the classical evaluation methods. For that purpose, we use a common test dataset allocated in the server. We show the evaluation metrics (loss and accuracy in this case) in each round of learning both in local models and global updated model. We show the behaviour of this evaluation methodology as follows.

[3]:

```
test_data = np.reshape(test_data, (test_data.shape[0], test_data.shape[1],␣
↪test_data.shape[2],1))
federated_government.run_rounds(1, test_data, test_labels)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x10a7d2d90>: [921.8931274414062, 0.19802500307559967]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x10a7ae590>: [208.905517578125, 0.5451250076293945]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x10a7ae450>: [528.586181640625, 0.30527499318122864]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x10a754610>: [264.839599609375, 0.5363749861717224]
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x10a74ac10>: [134.1905975341797, 0.6036249995231628]
Global model test performance : [29.60788345336914, 0.6252250075340271]
```

This methodology is the simplest and shows both local and global models. The problem with this methodology is that the local evaluation metrics are biased by the distribution of test set data. That is, the performance of the local models is not properly represented when using a non-iid scenario (see Federated Sampling) because the distribution of training data for each client is different from the data we test on. For that reason, we propose the following evaluation methodology.

1.2 Evaluation methodology 2: Global test dataset and local test datasets

In this evaluation methodology we consider that there is, as in the previous one, a global test dataset and that each client has a local test dataset according to the distribution of their training data. Hence, in each round we show the evaluation metrics of each client on their local and the global test. This evaluation methodology is more complete as it shows the performance of the local FL models in the global and local distribution of the data, which gives as more information.

First, we split each client's data in train and test partitions. You can find this method in Federated Operation.

[4]:
```
shfl.private.federated_operation.split_train_test(federated_data)
```

After that, each client owns a training set which uses for training the local learning model and a test set which uses to evaluate.

We are now ready to show the behaviour of this evaluation methodology.

[5]:
```
#We restart federated goverment
federated_government = shfl.federated_government.
↪FederatedGovernment(model_builder, federated_data, aggregator)
```

```
test_data = np.reshape(test_data, (test_data.shape[0], test_data.shape[1],␣
↪test_data.shape[2],1))
federated_government.run_rounds(1, test_data, test_labels)
```

```
Accuracy round 0
Performance client <shfl.private.federated_operation.FederatedDataNode object at
0x10a7d2d90>: Global test: [953.0712280273438, 0.19584999978542328], Local test:
[0.026163773611187935, 0.9906250238418579]
Performance client <shfl.private.federated_operation.FederatedDataNode object at
0x10a7ae590>: Global test: [282.0575256347656, 0.5266000032424927], Local test:
[0.2517445385456085, 0.9401041865348816]
Performance client <shfl.private.federated_operation.FederatedDataNode object at
0x10a7ae450>: Global test: [591.5661010742188, 0.2951500117778778], Local test:
[0.05175008624792099, 0.9788273572921753]
Performance client <shfl.private.federated_operation.FederatedDataNode object at
0x10a754610>: Global test: [306.59112548828125, 0.5108000040054321], Local test:
[0.348062127828598, 0.8961303234100342]
Performance client <shfl.private.federated_operation.FederatedDataNode object at
0x10a74ac10>: Global test: [125.78720092773438, 0.5453000068664551], Local test:
[0.7838365435600281, 0.7455470561981201]
Global model test performance : [62.16505813598633, 0.435824990272522]
```

We appreciate the significance of this new evaluation methodology in the output produced. For example, the first client performed the worst in the global test while it was the best in its local test. This indicates that the data distribution of this client is most likely very poor compared to the global data distribution, for example, consisting of only two classes. This produces a really good local learning model in just one round of learning, being a simpler problem but with a very low global test performance.

This highlights the strenght of using specific evaluation methodologies in FL, especially when the distribution of data among clients follows a non-IID distribution (see Federated Sampling).

```
[ ]:
```