federated_learning_attack_simulation_poisoning_data

June 18, 2020

1 Federated learning: Simulation of data poisoning attack

In this notebook provide a simulation of a simple data poisoning federated attack. We use a simple first approach which consists of shuffling the training labels of some number of clients which become adversarial ones.

The aim of this notebook is to present the class `FederatedDataAttack` implemented in federated_attack.py whose goal is to implement any attack on the federated data. For more information about basic federated learning concepts refer to the Basic Concepts Notebook.

For the simulation we use Emnist Digits dataset.

[1]:
```
from shfl.data_base import Emnist
from shfl.data_distribution import NonIidDataDistribution
import numpy as np
import random


database = Emnist()
train_data, train_labels, test_data, test_labels = database.load_data()
```

After that, we distribute the data among the client nodes using a non-iid distribution over 10% of the data.

[2]:
```
noniid_distribution = NonIidDataDistribution(database)
federated_data, test_data, test_labels = noniid_distribution.
 ↪get_federated_data(num_nodes=20, percent=10)
```

At this point, we are ready to apply some data attack to some nodes. For this simulation, we choose to apply data poisoning to the 20% of the nodes. For that purpose, we implement the interface `FederatedTransformation` with a shuffling of the training labels of `federated_data` and create `FederatedPoisoninigDataAttack`, which implements `FederatedDataAttack` with a data poisoning in a certain percentage of the nodes.

[3]:
```
from shfl.private.federated_operation import FederatedTransformation
from shfl.private.federated_attack import FederatedDataAttack

random.seed(123)

class ShuffleNode(FederatedTransformation):
```

```
    def apply(self, labeled_data):
        random.shuffle(labeled_data.label)

class FederatedPoisoningDataAttack(FederatedDataAttack):
    def __init__(self, percentage):
        super().__init__()
        self._percentage = percentage
        self._adversaries = []

    @property
    def adversaries(self):
        return self._adversaries

    def apply_attack(self, federated_data):
        num_nodes = federated_data.num_nodes()
        list_nodes = np.arange(num_nodes)
        self._adversaries = random.sample(list(list_nodes), k=int(self.
 ↪_percentage / 100 * num_nodes))
        boolean_adversaries = [1 if x in self._adversaries else 0 for x in␣
 ↪list_nodes]

        for node, boolean in zip(federated_data, boolean_adversaries):
            if boolean:
                node.apply_data_transformation(ShuffleNode())
```

We create a `FederatedPoisoningDataAttack` object with percentage set to 20% and apply the attack over `federated_data`.

[4]:
```
simple_attack = FederatedPoisoningDataAttack(percentage=20)
simple_attack.apply_attack(federated_data = federated_data)
```

We can get the adversarial nodes in order to show the attack applied.

[5]:
```
adversarial_nodes = simple_attack.adversaries
adversarial_nodes
```

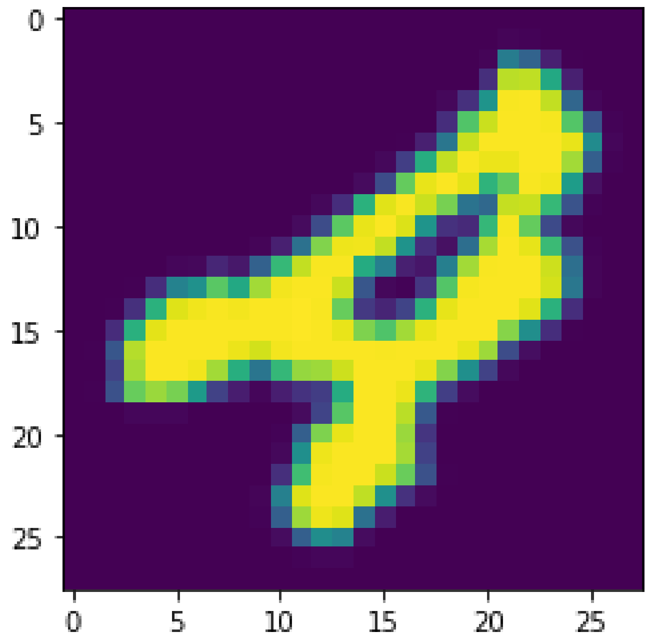
[5]:
```
[1, 8, 2, 13]
```

In order to show the effect of the attack, we select one adversarial client and an index position and show the data and the label associated with this image. We change data access protection (see FederatedData) in order to access the data. Due to the nature of the data poisoning (random shuffle) it is possible that for some specific data the label does match, but in most cases it does not.

[6]:
```
import matplotlib.pyplot as plt
from shfl.private.data import UnprotectedAccess

adversarial_index = 0
```

```
data_index = 10

federated_data.configure_data_access(UnprotectedAccess())

plt.imshow(federated_data[adversarial_nodes[adversarial_index]].query().
 ↪data[data_index])
print(federated_data[adversarial_index].query().label[data_index])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```



At this point, we can train a FL model among these clients (adversarial and regular ones) using a specific aggregation operator. For more information please visit Basic Concepts Notebook.

[ ]:

```
```