federated_models_logistic_regression

June 18, 2020

1 Federated learning: Classification by logistic regression

We show how to set up a Federated classification experiment using a Logistic Regression model. Results from the Federated Learning are compared to the (non-federated) centralized learning. Moreover, we also show how the addition of Differential Privacy affects the performance of the Federated model. In the present examples, we will generate synthetic data for the classification task. In particular, we will start from a two-dimensional case, since with only two features we are able to easily plot the samples and the decision boundaries computed by the classifier. After that, we will repeat the experiment by adding more features and classes to the synthetic database.

1.1 Two-features case:

We generate the data using the `make_classification` function from scikit-learn:

[1]:
```
import shfl
from shfl.data_base.data_base import LabeledDatabase
from sklearn.datasets import make_classification
import numpy as np
from shfl.private.reproducibility import Reproducibility

# Comment to turn off reproducibility:
Reproducibility(1234)

# Create database:
n_features = 2
n_classes = 3
n_samples = 500
data, labels = make_classification(
    n_samples=n_samples, n_features=n_features, n_informative=2,
    n_redundant=0, n_repeated=0, n_classes=n_classes,
    n_clusters_per_class=1, weights=None, flip_y=0.1, class_sep=0.4,␣
 ↪random_state=1234)
database = LabeledDatabase(data, labels)

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

print("Shape of train and test data: " + str(train_data.shape) + str(test_data.
 ↪shape))
```

```
print("Shape of train and test labels: " + str(train_labels.shape) +␣
↪str(test_labels.shape))
print(train_data[0,:])
```

```
Shape of train and test data: (400, 2)(100, 2)
Shape of train and test labels: (400,)(100,)
[-1.12355796 -1.1096599 ]
```

As mentioned, in this two-features case it is beneficial to visualize the results. For that purpose, we will use the following function:

[2]:
```
import matplotlib.pyplot as plt

def plot_2D_decision_boundary(model, data, labels, title=None):
    # Step size of the mesh. Smaller it is, better the quality
    h = .02
    # Color map
    cmap = plt.cm.Set1

    # Plot the decision boundary. For that, we will assign a color to each
    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Obtain labels for each point in mesh. Use last trained model.
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    fig, ax = plt.subplots(figsize=(9,6))
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=cmap,
               alpha=0.6,
               aspect='auto', origin='lower')
    # Plot data:
    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap=cmap, s=40, marker='o')

    plt.title(title, fontsize=18)
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('Feature 1', fontsize=18)
    plt.ylabel('Feature 2', fontsize=18)
    plt.tick_params(labelsize=15)
```

Running the Federated model. After defining the data, we are ready to run our model in a federated configuration. We distribute the data over the nodes, assuming the data is IID. Next, we define the aggregation of the federated outputs to be the *average* of the clients' models.

The *Sherpa.FL* framework offers support for the Logistic Regression model from scikit-learn. The user must specify in advance the number of features and the target classes: the assumption for this Federated Logistic Regression example is that each client's data possesses at least one sample of each class. Otherwise, each node might train a different classification problem, and it would be problematic to aggregate the global model. Setting model's state parameter to `warm_start:True` tells the clients to restart the training from the Federated round update. For assessing the performance, we compute the *Balanced Accuracy* and the *Kohen Kappa* scores (see metrics):

[3]:
```
from shfl.model.logistic_regression_model import LogisticRegressionModel

# Distribute data over the nodes:
iid_distribution = shfl.data_distribution.IidDataDistribution(database)
n_clients = 5
federated_data, test_data, test_labels = iid_distribution.
 ↪get_federated_data(num_nodes=n_clients, percent=100)
aggregator = shfl.federated_aggregator.FedAvgAggregator()

# Define the model:
classes = np.unique(train_labels)
def model_builder():
    model = LogisticRegressionModel(n_features=n_features, classes=classes,␣
 ↪model_inputs={'warm_start':True})
    return model

# Run the Federated experiment:
federated_government = shfl.federated_government.
 ↪FederatedGovernment(model_builder, federated_data, aggregator)
federated_government.run_rounds(n=2, test_data=test_data,␣
 ↪test_label=test_labels)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7d0e90>: (0.5694715007215007, 0.355612168439982)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5290>: (0.5937229437229438, 0.38732815301852963)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5410>: (0.5625270562770562, 0.34308748880262774)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5550>: (0.5486381673881674, 0.32442576189761296)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5690>: (0.48519119769119773, 0.2320433669628068)
Global model test performance : (0.6225018037518038, 0.4316482201615316)



Accuracy round 1
```

```
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7d0e90>: (0.5694715007215007, 0.355612168439982)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5290>: (0.5937229437229438, 0.38732815301852963)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5410>: (0.5625270562770562, 0.34308748880262774)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5550>: (0.5486381673881674, 0.32442576189761296)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5690>: (0.48519119769119773, 0.2320433669628068)
Global model test performance : (0.6225018037518038, 0.4316482201615316)
```

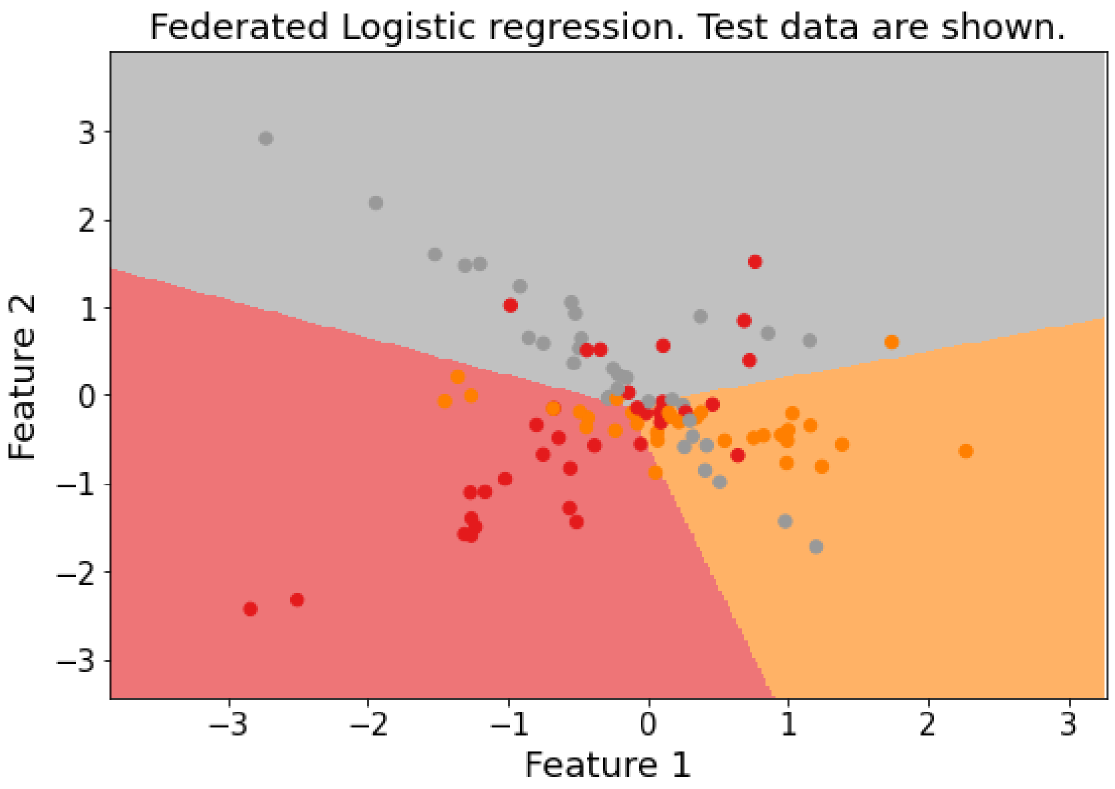
It can be observed that the performance of the *Federated Global model* is in general superior with respect to the performance of each node, thus the federated learning approach proves to be beneficial. Moreover, since no or little performance difference is observed between the federated rounds, we can conclude that the classification problem converges very early for this setting and no further rounds are required. This might be due to IID nature of clients' data when performing classification: each node gets a representative chunk of data and thus the local models are similar.

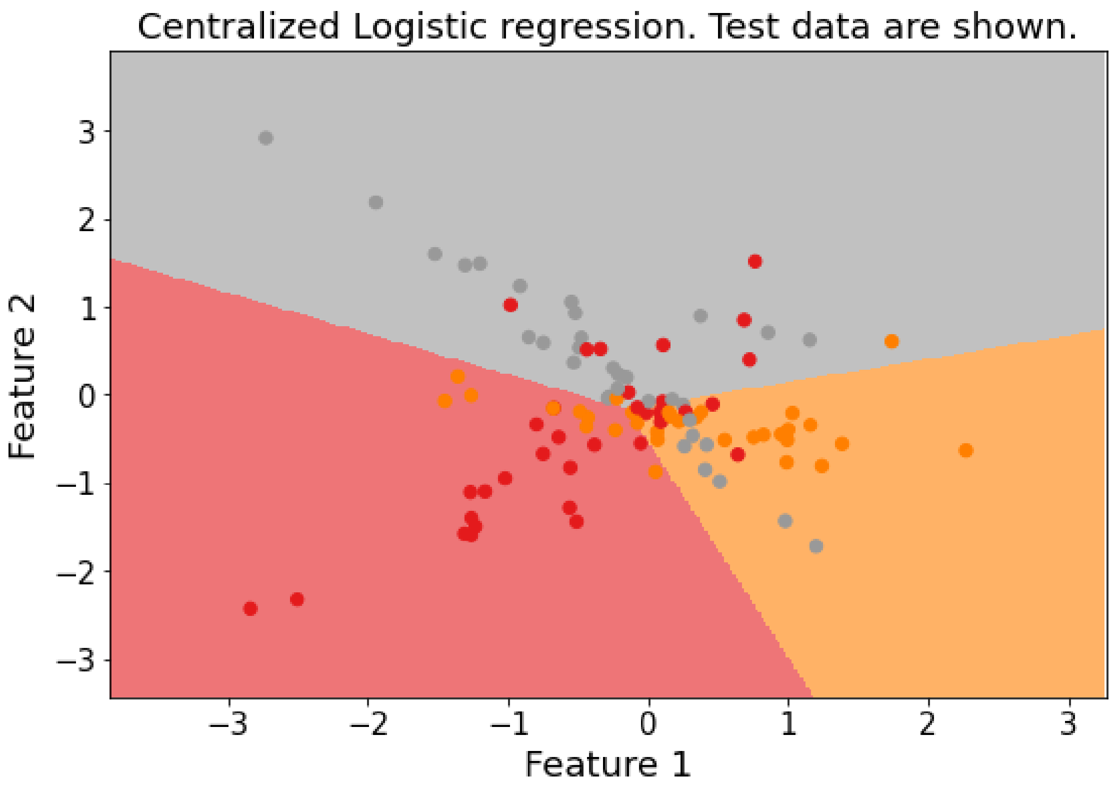
Comparison to centralized training. The performance of *Federated Global model* is *comparable* to the performance of the model trained on centralized data, and it produces similar decision boundaries:

[4]:
```python
# Train model on centralized data for comparison:
model_centralized = LogisticRegressionModel(n_features=n_features,␣
↪classes=classes)
model_centralized.train(train_data, train_labels)
print('Centralized test performance: ' + str(model_centralized.
↪evaluate(test_data, test_labels)))

# Plot decision boundaries and test data for Federated and (non-Federated)␣
↪centralized cases:
if n_features == 2:
    plot_2D_decision_boundary(federated_government.global_model, test_data,␣
↪test_labels, title = "Federated Logistic regression. Test data are shown.")
    plot_2D_decision_boundary(model_centralized._model, test_data, test_labels,␣
↪title = "Centralized Logistic regression. Test data are shown.")
```

```
Centralized test performance: (0.6326028138528138, 0.44635642675445153)
```


Federated Logistic regression. Test data are shown.
Feature 1
Feature 2
−3
−2
−1
0
1
2
3


Centralized Logistic regression. Test data are shown.
Feature 1
Feature 2
−3
−2
−1
0
1
2
3

Adding Differential Privacy to the Federated model. We want to assess the impact of Differential Privacy (see The Algorithmic Foundations of Differential Privacy, Section 3.3) on the Federated model's performance. In particular, we will use the Laplace mechanism (see also the corresponding Laplace mechanism notebook). The noise added has to be of the order of the sensitivity of the model's output, i.e. the model parameters of our logistic regression. In general, the sensitivity of a Machine Learning model is difficult to compute (for the Logistic Regression case, refer to Privacy-preserving logistic regression). An alternative strategy may be to estimate the sensitivity through a *sampling* procedure (e.g. see Rubinstein 2017 and how to use the tools provided by Sherpa.FL in the Linear Regression Notebook). However, be advised that this would guarantee the *weaker* property of *random* differential privacy. This approach is convenient since allows for the sensitivity estimation of an arbitrary model or a black-box computer function. The *Sherpa.FL* framework provides this functionality in the class `SensitivitySampler`.

We need to specify a distribution of the data to sample from, and this in general requires previous knowledge and/or model assumptions. In order not make any specific assumption on the distribution of the dataset, we can choose a *uniform* distribution. To the end, we define our class of `ProbabilityDistribution` that uniformly samples over a data-frame. We could sample using the train data. However, since in a real case the train data would be the actual clients' data, we wouldn't have access to it. Thus we generate another synthetic dataset for sampling (in the real case, this could be some public database we can access to):

[5]:
```
class UniformDistribution(shfl.differential_privacy.ProbabilityDistribution):
    """
    Implement Uniform sampling over the data
    """
    def __init__(self, sample_data):
        self._sample_data = sample_data

    def sample(self, sample_size):
        row_indices = np.random.randint(low=0, high=self._sample_data.shape[0],␣
 ↪size=sample_size, dtype='l')

        return self._sample_data[row_indices, :]

# Create sampling database:
n_samples = 150
sampling_data, sampling_labels = make_classification(
    n_samples=n_samples, n_features=n_features, n_informative=2,
    n_redundant=0, n_repeated=0, n_classes=n_classes,
    n_clusters_per_class=1, weights=None, flip_y=0.1, class_sep=0.1)
sample_data = np.hstack((sampling_data, sampling_labels.reshape(-1,1)))
```

The class `SensitivitySampler` implements the sampling given a *query*, i.e. the learning model itself in this case. We only need to add the method `get` to our model since it is required by the class `SensitivitySampler`: it simply trains the model on the input data and outputs the trained parameters. We choose the sensitivity norm to be the L_1 norm and we apply the sampling. The value of the sensitivity depends on the number of samples `n`: the more samples we perform, the more accurate the sensitivity. Indeed, increasing the number of samples `n`, the sensitivity gets more accurate and typically decreases. Note that the sampling could be quite costly, since at each time, the query (i.e. the model, in this case) is called.

[6]:
```
from shfl.differential_privacy import SensitivitySampler
from shfl.differential_privacy import L1SensitivityNorm

class LogisticRegressionSample(LogisticRegressionModel):

    def get(self, data_array):
        data = data_array[:, 0:-1]
        labels = data_array[:, -1]
        train_model = self.train(data, labels)

        return self.get_model_params()

distribution = UniformDistribution(sample_data)
sampler = SensitivitySampler()

n_samples = 200
max_sensitivity, mean_sensitivity = sampler.sample_sensitivity(
    LogisticRegressionSample(n_features=n_features, classes=classes),
    L1SensitivityNorm(), distribution, n=n_samples, gamma=0.05)
print("Max sensitivity from sampling: " + str(max_sensitivity))
print("Mean sensitivity from sampling: " + str(mean_sensitivity))
```

```
Max sensitivity from sampling: 0.5347092943796508
Mean sensitivity from sampling: 0.09472497472518714
```

Once obtained the model's estimated sensitivity, we fix the ϵ privacy budget and we can run the privacy-preserving Federated experiment:

[7]:
```
from shfl.differential_privacy import LaplaceMechanism

params_access_definition = LaplaceMechanism(sensitivity=max_sensitivity,␣
 ↪epsilon=0.5)
federated_governmentDP = shfl.federated_government.FederatedGovernment(
    model_builder, federated_data, aggregator,␣
 ↪model_params_access=params_access_definition)

federated_governmentDP.run_rounds(n=2, test_data=test_data,␣
 ↪test_label=test_labels)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7d0e90>: (0.5694715007215007, 0.355612168439982)
Test performance client <shfl.private.federated_operation.FederatedDataNode
```

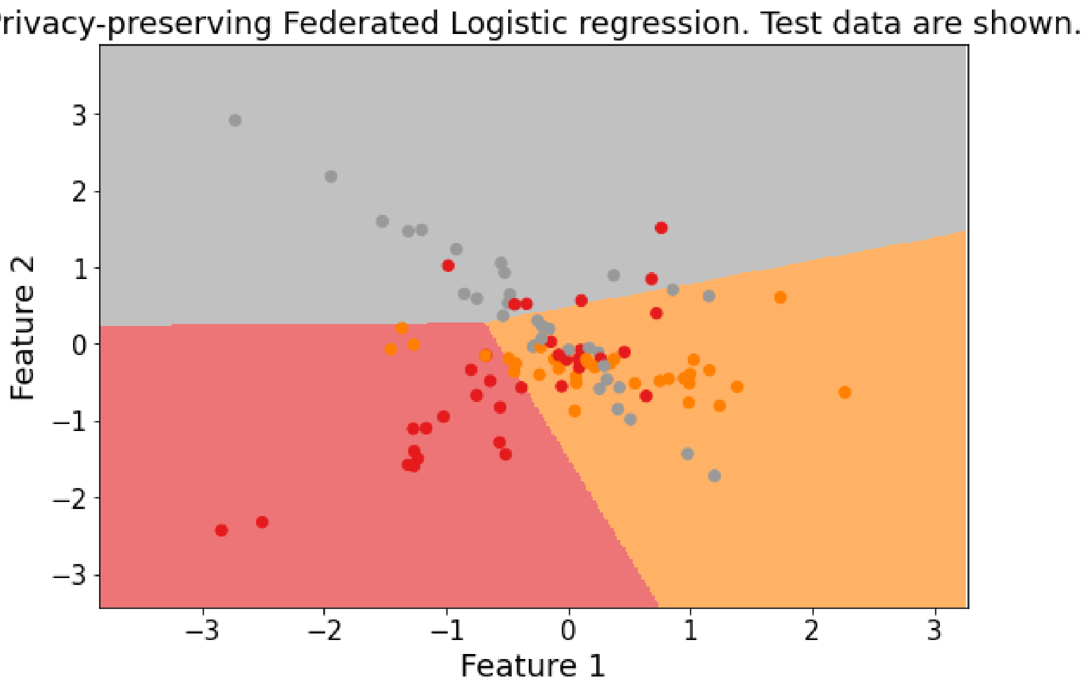
```
object at 0x13d7f5290>: (0.5937229437229438, 0.38732815301852963)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5410>: (0.5625270562770562, 0.34308748880262774)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5550>: (0.5486381673881674, 0.32442576189761296)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5690>: (0.48519119769119773, 0.2320433669628068)
Global model test performance : (0.564222582972583, 0.34966727162734423)



Accuracy round 1
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7d0e90>: (0.5694715007215007, 0.355612168439982)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5290>: (0.5937229437229438, 0.38732815301852963)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5410>: (0.5625270562770562, 0.34308748880262774)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5550>: (0.5486381673881674, 0.32442576189761296)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13d7f5690>: (0.48519119769119773, 0.2320433669628068)
Global model test performance : (0.5845689033189033, 0.3744414655942806)
```

As one could expect, the addition of random noise slightly alters the solution, but the model is still comparable to the non-private federated case:

[8]:
```
# Plot decision boundaries and test data for Privacy-preserving Federated case:
if n_features == 2:
    plot_2D_decision_boundary(federated_governmentDP.global_model, test_data,␣
 ↪test_labels,
                              title = "Privacy-preserving Federated Logistic␣
 ↪regression. Test data are shown.")
```



Remark 1: In this case we only run a few Federated rounds. However, in general one should make sure not to exceed a fixed privacy budget (see how to achieve that in *Sherpa.FL* in the Composition concepts notebook).
Remark 2: It is worth mentioning that the above results cannot be considered as general. Some factors that considerably influence the classification problem are, for example, the training dataset, the model type, and the Differential Privacy mechanism used. In fact, the classification problem itself depends on the training data (number of features, whether the classes are separable or not etc.). We strongly encourage the user to play with the values for generating the database (`n_features, n_classes, n_samples, class_sep ...` see their meaning), or try different classification datasets, since convergence and accuracy of local and global models can be strongly affected. Moreover, even without changing the dataset, by running the present experiment multiple times (you need to comment the *Reproducibility* command), it is observed that the *Federated Global model* may also exhibit slightly *better performance* when compared to the centralized model (we use `random_state` input for producing always the same dataset). Similarly, the privacy-preserving Federate model might exhibit even better performance compared to the non-private version. This depends on a) the performance metrics chosen and b) the idiosyncrasy of the specific classification considered here: a small modification to the model's coefficients may alter the class prediction for a few samples.

1.2 Case with more features and classes:

Below we present a more complex case, introducing more features and classes. When using more than 2 features, the figures are not plotted. Since the structure of the example is identical as above, the comments are not repeated:

```
[9]: # Create database:
n_features = 11
n_classes = 5
n_samples = 500
data, labels = make_classification(
    n_samples=n_samples, n_features=n_features, n_informative=4,
    n_redundant=0, n_repeated=0, n_classes=n_classes,
    n_clusters_per_class=2, weights=None, flip_y=0.1, class_sep=0.1,␣
 ↪random_state=123)
database = LabeledDatabase(data, labels)

train_data, train_labels, test_data, test_labels = database.load_data()
print("Shape of train and test data: " + str(train_data.shape) + str(test_data.
 ↪shape))
print("Shape of train and test labels: " + str(train_labels.shape) +␣
 ↪str(test_labels.shape))
print(train_data[0,:])
```

```
Shape of train and test data: (400, 11)(100, 11)
Shape of train and test labels: (400,)(100,)
[ 0.10007035 -1.58463047  1.28801948 -0.21302602 -0.22013894  1.13834715
 -1.21398519 -0.39202921 -0.87540182 -0.49998474 -1.07807789]
```

Running the model in a Federated configuration.

```
[10]: iid_distribution = shfl.data_distribution.IidDataDistribution(database)
federated_data, test_data, test_labels = iid_distribution.
 ↪get_federated_data(num_nodes=5, percent=100)

classes = np.unique(train_labels)
def model_builder():
    model = LogisticRegressionModel(n_features=n_features, classes=classes,␣
 ↪model_inputs={'warm_start':True})
    return model

aggregator = shfl.federated_aggregator.FedAvgAggregator()

federated_government = shfl.federated_government.
 ↪FederatedGovernment(model_builder, federated_data, aggregator)
federated_government.run_rounds(n=2, test_data=test_data,␣
 ↪test_label=test_labels)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db689d0>: (0.1654970760233918, -0.046394351991931426)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68cd0>: (0.2240601503759398, 0.027431421446384108)
Test performance client <shfl.private.federated_operation.FederatedDataNode
```

```
object at 0x13db68950>: (0.18137009189640768, -0.024615769086592465)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68250>: (0.22289055973266497, 0.03021260723610597)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68310>: (0.24411027568922306, 0.05130445637248804)
Global model test performance : (0.2625730994152047, 0.07615480649188511)
```

```
Accuracy round 1
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db689d0>: (0.1654970760233918, -0.046394351991931426)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68cd0>: (0.2240601503759398, 0.027431421446384108)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68950>: (0.18137009189640768, -0.024615769086592465)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68250>: (0.22289055973266497, 0.03021260723610597)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68310>: (0.24411027568922306, 0.05130445637248804)
Global model test performance : (0.2625730994152047, 0.07615480649188511)
```

Comparison with Centralized training.

[11]:
```
# Train model on centralized data:
model_centralized = LogisticRegressionModel(n_features=n_features,␣
 ↪classes=classes)
model_centralized.train(train_data, train_labels)
if n_features == 2:
    plot_2D_decision_boundary(model_centralized._model, train_data,␣
 ↪train_labels, title = "Benchmark: Logistic regression using Centralized␣
 ↪data")
print('Centralized test performance: ' + str(model_centralized.
 ↪evaluate(test_data, test_labels)))
```

```
Centralized test performance: (0.25304928989139513, 0.0647212869435091)
```

Adding Differential Privacy to the Federated model.

[12]:
```
# Create sampling database:
n_samples = 500
sampling_data, sampling_labels = make_classification(
    n_samples=n_samples, n_features=n_features, n_informative=4,
    n_redundant=0, n_repeated=0, n_classes=n_classes,
    n_clusters_per_class=2, weights=None, flip_y=0.1, class_sep=0.1,␣
 ↪random_state=123)
```

```
sample_data = np.hstack((sampling_data, sampling_labels.reshape(-1,1)))
distribution = UniformDistribution(sample_data)

# Sample sensitivity:
n_samples = 300
max_sensitivity, mean_sensitivity = sampler.sample_sensitivity(
    LogisticRegressionSample(n_features=n_features, classes=classes),
    L1SensitivityNorm(), distribution, n=n_samples, gamma=0.05)
print("Max sensitivity from sampling: " + str(max_sensitivity))
print("Mean sensitivity from sampling: " + str(mean_sensitivity))
```

```
Max sensitivity from sampling: 0.9198337971901624
Mean sensitivity from sampling: 0.42652826621356754
```

[13]:
```
from shfl.differential_privacy import LaplaceMechanism

params_access_definition = LaplaceMechanism(sensitivity=max_sensitivity,␣
 ↪epsilon=0.5)
federated_governmentDP = shfl.federated_government.FederatedGovernment(
    model_builder, federated_data, aggregator,␣
 ↪model_params_access=params_access_definition)

federated_governmentDP.run_rounds(n=2, test_data=test_data,␣
 ↪test_label=test_labels)
```

```
Accuracy round 0
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db689d0>: (0.1654970760233918, -0.046394351991931426)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68cd0>: (0.2240601503759398, 0.027431421446384108)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68950>: (0.18137009189640768, -0.024615769086592465)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68250>: (0.22289055973266497, 0.03021260723610597)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68310>: (0.24411027568922306, 0.05130445637248804)
Global model test performance : (0.23659147869674185, 0.04714763024378177)



Accuracy round 1
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db689d0>: (0.1654970760233918, -0.046394351991931426)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68cd0>: (0.2240601503759398, 0.027431421446384108)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68950>: (0.18137009189640768, -0.024615769086592465)
```

```
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68250>: (0.22289055973266497, 0.03021260723610597)
Test performance client <shfl.private.federated_operation.FederatedDataNode
object at 0x13db68310>: (0.24411027568922306, 0.05130445637248804)
Global model test performance : (0.20676691729323307, 0.006539235412475031)
```