Differential Privacy in Machine Learning

Sumit Agrawal

Akshay Bhandary

Abstract

Machine learning algorithms are widely used nowadays, and most of the training datasets contain the sensitive information of the individuals to train the model. Most of the training datasets contain the sensitive information of the individuals to train the model. Commonly, a technique based on the neural network is often used to get the high efficiency of the model. To train such a model, we need a large dataset that may contain sensitive information, and this dataset could be released publicly. However, the model should not uncover the confidential data of individuals. Addressing this scenario, we create a new algorithm technique that will train the model under a privacy budget so that the dataset is encrypted. The trade-off is that it will affect the model accuracy, model quality, and training time complexity at a manageable cost.

Acknowledgments

The writers of this paper stretch out their gratitude to Professor Ming-Hwa Wang for his help in deciding on the idea for this task. Moreover, the Professor's instruction on the basics of data mining gave us the capacity to comprehend and execute differential privacy.

Table of Contents

Introduction

Objective

The goal of differential privacy in Machine Learning is to encrypt the dataset so that the attacker cannot ascertain information from the model's output about any observations that could be used to identify the individual whose private information was used to train the model. Also, the efficiency of the model should not have any significant impact.

What is the problem

Machine Learning algorithms are used to understand and form relationships, and patterns like "patients who smoke are more likely to have cancer" in large datasets. However, these algorithms also learn specific training examples by default, which, if not ignored, can lead to information about the training set being revealed once the model is deployed for public use. Researchers have proposed and deployed different approaches to protect the privacy of the dataset in learning algorithms. Unfortunately, all these approaches are vulnerable to composition attacks. For instance, Netflix came up with a competition called Netflix Prize to improve their movie recommendation algorithm by crowdsourcing it. To train a model, one needs data, which Netflix had graciously provided but had also "anonymized" the data that had been shared. However, just 16 days later, researchers from the University of Texas were able to identify some of the users by matching the data provided by Netflix and data from other sites like IMDB [1].

Another problem that comes with releasing data, even after the data is "anonymized," is that there is a limit to the anonymization, which can be easily overcome by combining data from a different source to uniquely identify patterns that can then be used to identify individuals and ascertain private information of individuals—for example, the Massachusetts mayor's medical records. In the mid-1990s, Massachusetts Group Insurance Commission released

"anonymized" data about state employee hospital visits to help researchers. However, a graduate student was able to identify Massachusetts Governor's entire medical record by combining data from voter rolls of the state of Cambridge, which included name, address, birth date, etc. She had then sent the Governor's health record, including diagnosis and prescriptions, to his office [2].

Why is this problem related to this class?

Data Mining is the process of extracting and discovering patterns in large datasets using advanced learning algorithms and machine learning models. Our project will try to classify handwritten digits from the MNIST database using a machine learning algorithm known as CNN, an advanced form of a Neural Network taught in class. Optimizing algorithms such as Gradient Descent are also used in conjunction with CNN for many applications; we will use it to address security and privacy concerns. Our topic (DP in ML) concerns the security of the databases used to train such complex machine learning models and is currently being researched and worked on. Therefore, we will be using many concepts and challenges taught as part of the syllabus for this class.

Why other approaches are not good.

Other approaches to address security and privacy concerns revolve around anonymizing a database. For this, algorithms have developed such as k-anonymity and l-diversity. However, there are a couple of problems with such approaches.

Models like k-anonymity and other related notions attempt to offer security against identification attacks by suppressing some identifying attributes. But they do so by learning and retaining partial information about different attributes or dimensions, which is a cause of concern since this renders them more open to inference attacks. Secondly, such models are

only feasible when the databases are smaller from a dimensionality point of view. Some research has been conducted which shows that in cases where the database has high dimensionality, the loss of information that occurs to provide 2-anonymity may not be acceptable from a data mining point of view. Thirdly, algorithms have been developed that need only statistical information about data to make machine learning models. Such algorithms are also designed in such a way to be executed on changed data with much stronger privacy guarantees than by using algorithms like k-anonymity [3]. Finally, some models have been made to overcome the security concern by treating the whole training process as a black box and only working on the final model produced because of the black box training process. However, in cases where the parameters of the data are more tightly integrated, or the data has more intricate and interdependent relationships between attributes in the data, it will cause the model to be trained in a wrong way once the noise is added to the data.

Why we think our approach is better

In our Approach, the effect of making an arbitrary single substitution in the database is minimal; therefore, it will impact the output of the model up to a small extent but ensures that the privacy of the confidential information is maintained. To elaborate, when white noise is added to the training dataset, the accuracy of the model decreases by a margin of 6 to 10%; however, the trade-off is that even if the adversary has a similar dataset from different sources, they will not be able to recognize any pattern when the differentially private dataset is released. Hence, it ensures that the information of the individuals in the differentially private dataset is secured.

Area or scope of the investigation

We will be trying to implement differential privacy by adding noise to the dataset and then evaluating the model's accuracy when trained by differentially private data and trained by unchanged data. We will also be assessing the loss of information that may occur for a differentially private dataset.

Theoretical bases and literature review

Definition of Problem:

The problem we face is ensuring the privacy of sensitive information in datasets used to build machine learning models. Our project revolves around solving the privacy and security issues raised while creating/building machine learning models to solve more significant problems. Although there have been different approaches to solving these issues, most of them have severe shortcomings concerning the feasibility of the solution and trade-off with privacy and accuracy of the model.

Related Research to solve the problem

Other approaches try to anonymize data by removing or suppressing or modifying attributes that could be used to identify individuals in the dataset. The most common is the k-anonymity model proposed by L Sweeney. Also known as hiding in the crowd, the main idea behind k-anonymity is to pool an individual's data with a larger group, thus masking the individual's identity [4].

Advantages/disadvantages of those research

The significant advantage of algorithms like k-anonymity is that re-identifying records become nearly impossible. However, it comes at the cost of considerable information loss and hence can be only used for huge datasets to make it feasible from both a commercial and data mining point of view. Also, this method was developed in 1998; data from sources like a social network, which is much more complicated than relational data, makes it even more challenging.

Our solution to solve this problem:

To solve the problem of ensuring the privacy of sensitive information in datasets used to build machine learning models, we are using the concept of differential privacy to make it exponentially difficult for any unauthorized person to identify any individuals or find patterns in the dataset that could be used with other datasets to identify individuals, thus ensuring privacy.

We will be doing so by adding noise randomly in the dataset that will deter any adversary from inferring any information about any individual while also taking care of the trade-off between accuracy and privacy to make sure that the model will be able to classify data with equal accuracy with the perturbed data as with collected data.

Where our solution different from others

Other solutions to solve the problem involve changing databases by removing attributes from the databases that could identify individuals uniquely or could be used to find patterns that could be used in conjunction with other databases to identify individuals; such attempts are called composition attacks. However, the trade-off between accuracy and privacy is not as low as necessary to make it viable from a data mining point of view.

Combined with the fact that removing attributes from the database also affects accuracy adversely since important information may be omitted during the learning/training phase of the model, it makes other solutions less effective as compared to ours since, in our project, the loss of information and accuracy are both at a negligible level.

Why our solution is better

The method of adding noise randomly is known as differential privacy. The goal of differential privacy in a model is to ensure that the dataset is indistinguishable from whether a

single observation is present or not. As an added benefit, differential privacy has several properties like composability and group privacy, making it a viable alternative from both a commercial standpoint and a data mining point of view. Composability states that if all small components of a mechanism are differentially private, then the sum of all parts, the whole mechanism, is also differentially private.

Our solution to the problem of ensuring privacy involves adding noise randomly to the database at a level that will make it difficult for an adversary to infer any knowledge from the database but at the same time not affect the model's training, learning, and accuracy by a large extent, thus making it viable and feasible from a data mining point of view.

Hypothesis

Single Hypothesis:

Using the idea behind differential privacy in machine learning, we will be adding noise to a dataset to significantly reduce the likelihood of re-identification of individuals or identification of patterns that could potentially be used to identify information about individuals in the dataset

Positive Hypothesis:

By adding noise, we hypothesize that the privacy of sensitive and confidential information of individuals on whose data the database has been made will be guaranteed to a much stronger degree than the previous approach using algorithms meant to anonymize data like k-anonymity.

Negative Hypothesis:

Although privacy will be guaranteed for individuals' private information, the caveat with this method is that the machine learning model will be slightly less accurate than if the model had been trained with non-perturbed data.

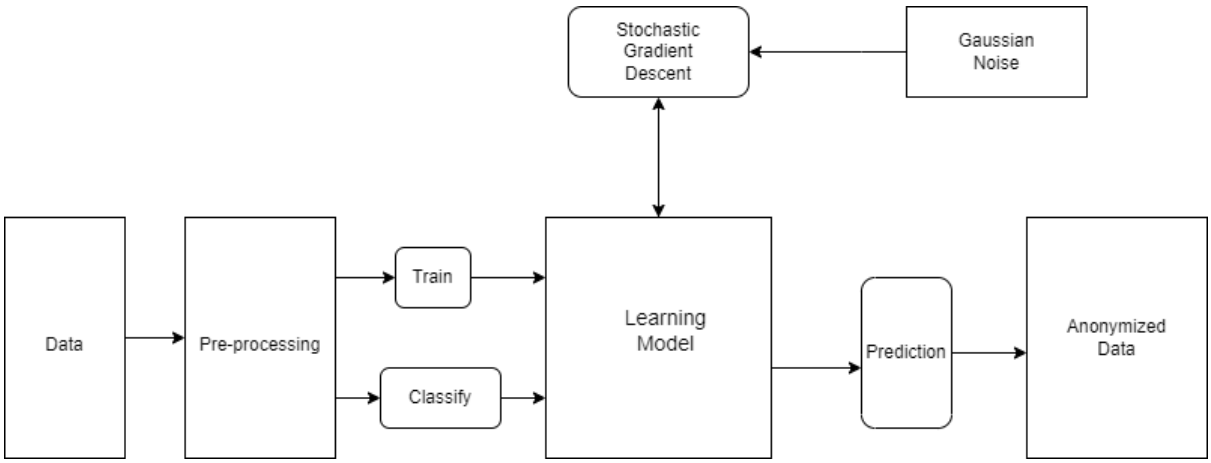
Methodology

How to generate/collect input data

We will be using the MNIST dataset to train the model to detect and classify the digits of handwritten numbers, which has 60 thousand grayscale images of handwritten single digits between 0 and 9, thus satisfying our requirement for a large dataset.

How to solve the problem

Algorithm Design



Our algorithm to solve the problem has the following steps:

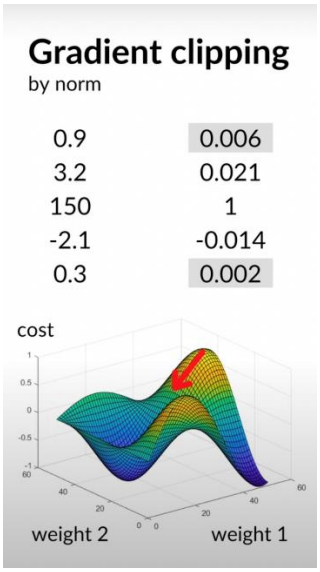
1. We will start by downloading and converting the MNIST datasets into two data loaders for training and testing using Pytorch.
2. Following this, we will train a convolution neural network to classify the dataset accordingly using the PyTorch library to build and train the model and implement the optimization algorithm such as Stochastic Gradient Descent.
3. Once the training is completed, we will test the model against the MNIST database and calculate the classification's accuracy.
4.

We first perform Gradient Clipping for the next phase, a technique used to prevent exploding gradients.

So, if the value of gradient **g** is greater than the hyperparameter **c**, then the value of the gradient is changed as below:

$\mathbf{g} \leftarrow c \cdot \mathbf{g}/\|\mathbf{g}\|$

$$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$$



5. After the gradient clipping process has been completed, we will add noise to the MNIST database. While calculating the noise to be added, we will be using the privacy budget (ϵ, epsilon) to ensure that the change in the model's output should not vary by a vast amount.
6. Finally, we will retrain the model using the changed database, evaluate the accuracy and loss of information, if any, and compare it with the previous training results.

Language Used

We will be using Python as the programming language to implement differential privacy. It has excellent support libraries, enabling us to implement algorithms such as Stochastic Gradient Descent and help train a Convolution Neural Network.

Tools Used

The tools we will be using are listed below:

1. Google Colab
2. Opacus Library
3. NumPy

4. Matplot
5. PyTorch

How to generate Output

Our project will have two outputs that will be used to verify differential privacy. The first will be an unchanged training dataset used to evaluate the accuracy of the model trained by current methodologies. The second will be changed differentially private dataset, which will be used to train the model and evaluate the impact on the accuracy of the model.

How to test against the hypothesis

Our hypothesis, to implement differential privacy, will be tested by checking the impact on accuracy and loss of information from a model trained using both a differentially private dataset and a nonperturbed dataset to see if the addition of noise randomly will affect the accuracy of the prediction/classification and if this trade-off will make it viable from a data mining standpoint.

Implementation

Code

```
!pip install opacus
import torch
from torchvision import datasets, transforms
import numpy as np
from opacus import PrivacyEngine
from tqdm import tqdm
import matplotlib.pyplot as plt
from PIL import Image
from numpy import vstack
from numpy import argmax
from sklearn.metrics import accuracy_score

train_loader = torch.utils.data.DataLoader(datasets.MNIST('../mnist', train=True, download=True,
                                                          transform=transforms.Compose(
                                                              [transforms.ToTensor(), transforms.Normalize((0.1307,),
                                                                                                           (
                                                                                                           0.3081,))]), ),
                                           batch_size=64, shuffle=True, num_workers=1, pin_memory=True)

test_loader = torch.utils.data.DataLoader(datasets.MNIST('../mnist', train=False,
                                                         transform=transforms.Compose(
                                                             [transforms.ToTensor(), transforms.Normalize((0.1307,),
                                                                                                          (
                                                                                                          0.3081,))]), ),
                                          batch_size=1024, shuffle=True, num_workers=1, pin_memory=True)
```

```
# Step 3: Creating a PyTorch Neural Network Classification Model and Optimizer
model = torch.nn.Sequential(torch.nn.Conv2d(1, 16, 8, 2, padding=3), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                            torch.nn.Conv2d(16, 32, 4, 2), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                            torch.nn.Flatten(),
                            torch.nn.Linear(32 * 4 * 4, 32), torch.nn.ReLU(), torch.nn.Linear(32, 10))

optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
```

```python
def train_notprivate(model, train_loader, optimizer, epoch, device):
    model.train()
    criterion = torch.nn.CrossEntropyLoss()
    losses = []
    for _batch_idx, (data, target) in enumerate(tqdm(train_loader)):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    print(
        f"Train Epoch: {epoch} \t"
        f"Loss: {np.mean(losses):.6f} ")

for epoch in range(1, 11):
    train_notprivate(model, train_loader, optimizer, epoch, device="cpu")

def evaluate_model(test_dl, model):
    predictions, actuals = list(), list()
    for i, (inputs, targets) in enumerate(test_dl):
      # evaluate the model on the test set
      yhat = model(inputs)
      # retrieve numpy array
      yhat = yhat.detach().numpy()
      actual = targets.numpy()
      # convert to class labels
      yhat = argmax(yhat, axis=1)
      # reshape for stacking
      actual = actual.reshape((len(actual), 1))
      yhat = yhat.reshape((len(yhat), 1))
      # store
      predictions.append(yhat)
      actuals.append(actual)
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc


 # prepare the data
train_dl, test_dl = train_loader, test_loader
print(len(train_dl.dataset), len(test_dl.dataset))
acc = evaluate_model(test_dl, model)
print('Accuracy: %.3f' % acc)
```

```python
DP_model = torch.nn.Sequential(torch.nn.Conv2d(1, 16, 8, 2, padding=3), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                            torch.nn.Conv2d(16, 32, 4, 2), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                            torch.nn.Flatten(),
                            torch.nn.Linear(32 * 4 * 4, 32), torch.nn.ReLU(), torch.nn.Linear(32, 10))

DP_optimizer = torch.optim.SGD(DP_model.parameters(), lr=0.05)

# Attaching a Differential Privacy Engine to the Optimizer
privacy_engine= PrivacyEngine()
N_model, N_optimizer, dataloader = privacy_engine.make_private(
    module=DP_model,
    optimizer=DP_optimizer,
    data_loader=train_loader,
    noise_multiplier=1.2,
    max_grad_norm=1.0,
    poisson_sampling=False,
)
```

```
def get_epsilon(self, delta):
  return self.accountant.get_epsilon(delta)
# Training the private model over multiple epochs
def train(model, train_loader, optimizer, epoch, device, delta):
    model.train()
    criterion = torch.nn.CrossEntropyLoss()
    losses = []
    for _batch_idx, (data, target) in enumerate(tqdm(train_loader)):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    epsilon=privacy_engine.get_epsilon(delta)
    print(
        f"Train Epoch: {epoch} \t"
        f"Loss: {np.mean(losses):.6f} "
        f"(ε = {epsilon:.2f}, δ = {delta})")

for epoch in range(1, 11):
    train(N_model, dataloader, N_optimizer, epoch, device="cpu", delta=1e-5)
```

```
acc = evaluate_model(test_loader, N_model)
print('Accuracy: %.3f' % acc)

dataloader.batch_size
examples = enumerate(dataloader)
batch_idx, (example_data, example_targets) = next(examples)
```

```
example_data.shape

fig = plt.figure()
for i in range(6):
  plt.subplot(2,3,i+1)
  plt.tight_layout()
  plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
  plt.title("Ground Truth: {}".format(example_targets[i]))
  plt.xticks([])
  plt.yticks([])
fig
```

```
print(N_model)
model_child = list(N_model.children())
```

```
import torch.nn as nn
no_of_layers=0
conv_layers=[]
for child in model_child:
  if type(child)==nn.Conv2d:
    no_of_layers+=1
    conv_layers.append(child)
  elif type(child)==nn.Sequential:
    for layer in child.children():
      if type(layer)==nn.Conv2d:
        no_of_layers+=1
        conv_layers.append(layer)
print(no_of_layers)
```

Design Document and flowchart

Our project that shows how differential privacy is used in the machine has been designed using the following steps:

1. First, we downloaded and converted the MNSIT dataset into two portions of approximately 85% meant for training and 15% for testing using the python library PyTorch to convert the dataset into dataloaders.
2. After converting and splitting the dataset, we trained a basic convolution neural network to classify the dataset accordingly using PyTorch to build and train the model. This is done to establish a base line for comparing against the model which will be trained with a differentially private data.
3. After completion of training, we tested the model against the MNIST database and calculated the classification's accuracy.
4. For the next phase of the project, we performed Gradient Clipping by norm to prevent exploding gradients. This was done because while using the Stochastic Gradient descent algorithm as the optimization algorithm for the CNN model, we have to make sure that the algorithm finds the global minima and does not move towards the local minima or any other direction.
5. Following that, we added noise to the MNIST data. This noise added, was calculated using the privacy budget (ϵ, epsilon) to make sure that the model's output should not vary by a vast amount
6. Finally, we retrained the model using the database to which noise was added, and then evaluated the accuracy and loss of information, then compared it with the previous training results to understand the efficacy of differential privacy in machine learning.

Data analysis and discussion

Output Generation

The output for the project will showcase the efficacy of differential privacy in machine learning by showing the following:

1. The accuracy of the model when trained with a normal dataset
2. Loss of Information incurred while training with a normal dataset
3. The accuracy of the model when trained with a differentially private dataset
4. Loss of Information incurred while training with a differentially private dataset
5. The images of handwritten digits before and after the addition of varying levels of noise

These outputs were generated by using inbuilt python functionalities and some libraries like NumPy, Sci-Kit Learn, and matplotlib.

Output Analysis

As can be seen from the below screenshot, the accuracy of the model without adding noise to the dataset is 98.8% and the loss of information at the end of training is 2%, which showcases the efficiency of the CNN model in normal usage.

```
100%|██████████| 938/938 [00:36<00:00, 25.60it/s]

Train Epoch: 1  Loss: 0.312060

100%|██████████| 938/938 [00:26<00:00, 35.92it/s]

Train Epoch: 2  Loss: 0.075240

100%|██████████| 938/938 [00:25<00:00, 36.17it/s]

Train Epoch: 3  Loss: 0.053376

100%|██████████| 938/938 [00:25<00:00, 36.17it/s]

Train Epoch: 4  Loss: 0.042650

100%|██████████| 938/938 [00:27<00:00, 34.58it/s]

Train Epoch: 5  Loss: 0.034689

100%|██████████| 938/938 [00:26<00:00, 36.02it/s]

Train Epoch: 6  Loss: 0.030651

100%|██████████| 938/938 [00:26<00:00, 34.97it/s]

Train Epoch: 7  Loss: 0.027995

100%|██████████| 938/938 [00:26<00:00, 35.79it/s]

Train Epoch: 8  Loss: 0.023730

100%|██████████| 938/938 [00:29<00:00, 32.01it/s]

Train Epoch: 9  Loss: 0.020594
```

```python
def evaluate_model(test_dl, model):
    predictions, actuals = list(), list()
    for i, (inputs, targets) in enumerate(test_dl):
      # evaluate the model on the test set
      yhat = model(inputs)
      # retrieve numpy array
      yhat = yhat.detach().numpy()
      actual = targets.numpy()
      # convert to class labels
      yhat = argmax(yhat, axis=1)
      # reshape for stacking
      actual = actual.reshape((len(actual), 1))
      yhat = yhat.reshape((len(yhat), 1))
      # store
      predictions.append(yhat)
      actuals.append(actual)
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc


# prepare the data
train_dl, test_dl = train_loader, test_loader
print(len(train_dl.dataset), len(test_dl.dataset))
acc = evaluate_model(test_dl, model)
print('Accuracy: %.3f' % acc)
```

```
60000 10000
Accuracy: 0.988
```

Next, we see the training of the same model with the addition of noise calculated using the privacy engine. Here we see that the loss of information is 20%, which is greater than the loss of information when the model was trained without noise. This is the trade-off that we have made with privacy, the addition of noise, accuracy, and loss of information.

```
100%|██████████| 938/938 [00:51<00:00, 18.11it/s]
Train Epoch: 1  Loss: 1.164523 (ε = 0.68, δ = 1e-05)
100%|██████████| 938/938 [00:42<00:00, 21.87it/s]
Train Epoch: 2  Loss: 0.572431 (ε = 0.69, δ = 1e-05)
100%|██████████| 938/938 [00:43<00:00, 21.37it/s]
Train Epoch: 3  Loss: 0.542119 (ε = 0.71, δ = 1e-05)
100%|██████████| 938/938 [00:41<00:00, 22.55it/s]
Train Epoch: 4  Loss: 0.507823 (ε = 0.72, δ = 1e-05)
100%|██████████| 938/938 [00:41<00:00, 22.48it/s]
Train Epoch: 5  Loss: 0.474783 (ε = 0.73, δ = 1e-05)
100%|██████████| 938/938 [00:41<00:00, 22.47it/s]
Train Epoch: 6  Loss: 0.451766 (ε = 0.75, δ = 1e-05)
100%|██████████| 938/938 [00:41<00:00, 22.47it/s]
Train Epoch: 7  Loss: 0.442711 (ε = 0.76, δ = 1e-05)
100%|██████████| 938/938 [00:42<00:00, 21.92it/s]
Train Epoch: 8  Loss: 0.419686 (ε = 0.78, δ = 1e-05)
100%|██████████| 938/938 [00:46<00:00, 20.10it/s]
Train Epoch: 9  Loss: 0.398222 (ε = 0.79, δ = 1e-05)
```

Here we can see that the accuracy of the model has dropped to 92% which is less than that when the model was trained on the original dataset, however, an accuracy of 92% makes the model viable from a data mining point of view

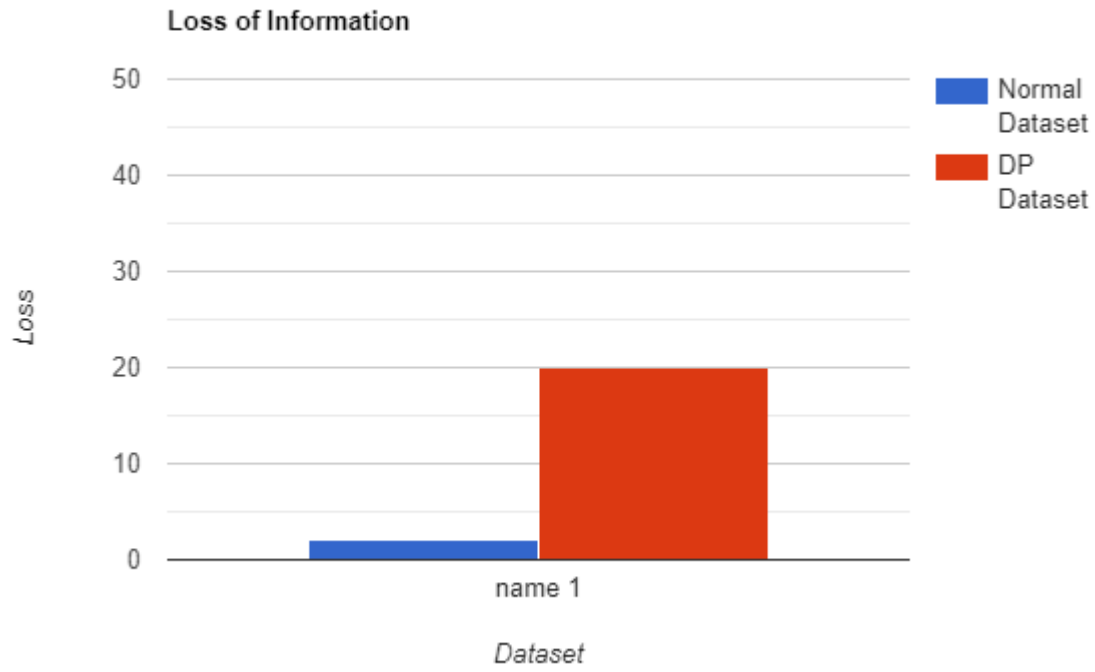
```
    acc = evaluate_model(test_loader, N_model)
    print('Accuracy: %.3f' % acc)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py:1033: UserWarning: Using a non-full backward hook when the forwa
This hook will be missing some grad_input. Please use register_full_backward_hook to get the documented behavior.
  warnings.warn("Using a non-full backward hook when the forward contains multiple autograd Nodes "

Accuracy: 0.934
```

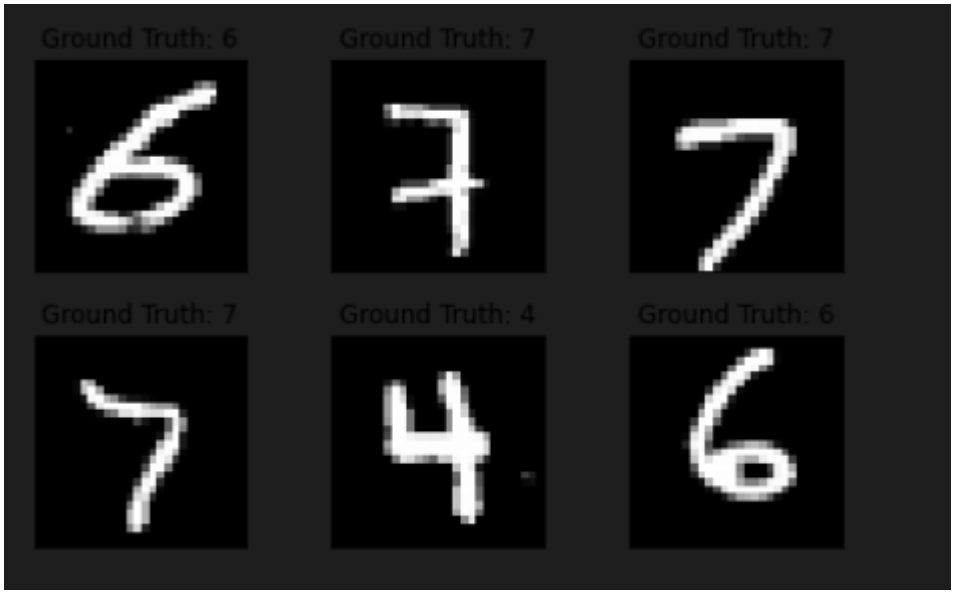
As can be seen from the graph below, the accuracy of the model when trained with a normal dataset and when trained with a dataset that is differentially private because of the addition of noise is relatively the same and thus makes the differential private model viable from a data mining and commercial point of view.

However, as can be seen from the below graph of loss of information in both a normal dataset and a differentially private dataset, the loss of information is greater,



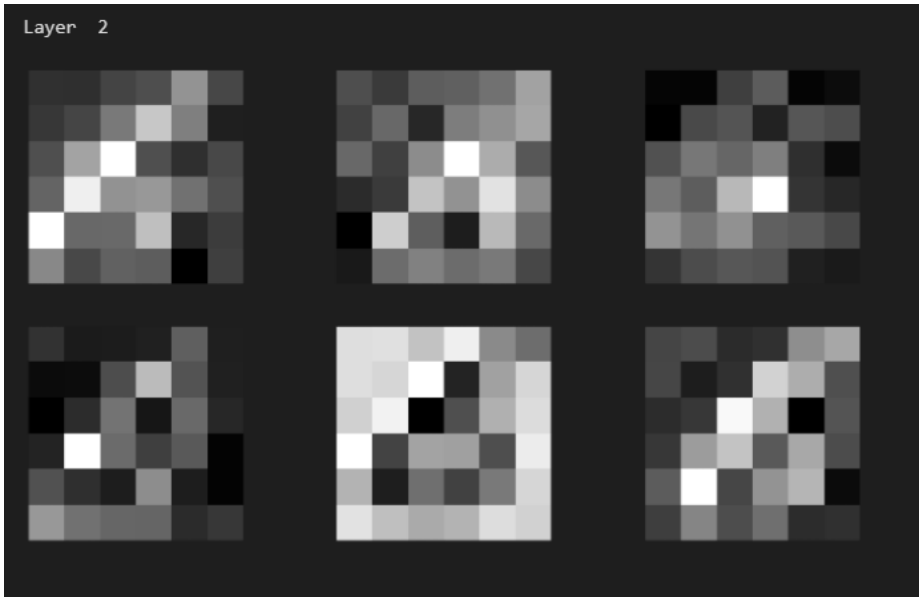
however, given the fact that the accuracy of the model is at a comparable level, and that the privacy of the information of individuals is guaranteed, the loss of information is a trade-off that is made necessary.

Compare Output against Hypothesis



As we can ascertain from the above screenshot, the ground truth that is the images of handwritten digits from the MNIST dataset are clear.

But, as can be seen from the images of the second layer of the CNN, the addition of noise makes the image too noisy and too blurry for human eyes to ascertain which number is being shown, thus proving that the addition of noise does impact the dataset which makes it exponentially difficult for an unauthorized person to ascertain any information that could lead to the identification of individuals from the differentially private dataset.



Along with that the output from the second layer of the CNN, also shows that the model can understand, learn, and classify digits accurately even with the addition of noise, thus proving the efficacy of differential privacy.

Conclusions and Recommendations

Summary and Conclusions

To summarise, what we have shown here is that it is indeed possible for a data scientist to create machine learning models that can learn general relationships between data without having to learn specific relationships like for example, a model that is built to learn to detect cancer will learn that people who smoke are more susceptible to get lung cancer or people who chew tobacco are a higher risk to get mouth cancer, without having access to specific details like Bob Barker is a chain smoker and therefore is more susceptible to get lung cancer. This is how the privacy of individuals will be saved.

Also, from a data scientist's point of view, we are more interested in the patterns that occur most often in a dataset, which closely mirrors the generalization that we need in machine learning and statistics and not the one-off relationships that could be found in a dataset, which is why this approach is realistic.

Recommendations for future studies

The first goal for future studies would be to try and reduce the loss of information on a differentially private dataset by using the privacy budget in an incremental way to reduce the loss of information.

However, the main goal of future studies for differential privacy in machine learning is not only to address privacy concerns on the dataset but also to develop a method wherein multiple people can combine their private inputs to compute a function without having to reveal their inputs to each other. This can be done using encryption and cryptographic mathematics. This will enable the different parties to work on the encrypted dataset to build

models with the caveat being that the only method to decrypt the dataset for any reason would be for all parties concerned to come together and agree to do so.

Appendices

Code

```
!pip install opacus

import torch

from torchvision import datasets, transforms

import numpy as np

from opacus import PrivacyEngine

from tqdm import tqdm

import matplotlib.pyplot as plt

from PIL import Image

from numpy import vstack

from numpy import argmax

from sklearn.metrics import accuracy_score

train_loader = torch.utils.data.DataLoader(datasets.MNIST('../mnist', train=True, download=True,

                                                          transform=transforms.Compose(

                                                              [transforms.ToTensor(), transforms.Normalize((0.1307,),

                                                                                                           (

                                                                                                               0.3081,)), ]), ),

                                           batch_size=64, shuffle=True, num_workers=1, pin_memory=True)


test_loader = torch.utils.data.DataLoader(datasets.MNIST('../mnist', train=False,

                                                         transform=transforms.Compose(

                                                             [transforms.ToTensor(), transforms.Normalize((0.1307,),

                                                                                                          (

                                                                                                              0.3081,)), ]), ),

                                          batch_size=1024, shuffle=True, num_workers=1, pin_memory=True)
```

```
# Step 3: Creating a PyTorch Neural Network Classification Model and Optimizer
model = torch.nn.Sequential(torch.nn.Conv2d(1, 16, 8, 2, padding=3), torch.nn.ReLU(),
torch.nn.MaxPool2d(2, 1),
                    torch.nn.Conv2d(16, 32, 4, 2), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                    torch.nn.Flatten(),
                    torch.nn.Linear(32 * 4 * 4, 32), torch.nn.ReLU(), torch.nn.Linear(32, 10))

optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
def train_notprivate(model, train_loader, optimizer, epoch, device):
    model.train()
    criterion = torch.nn.CrossEntropyLoss()
    losses = []
    for _batch_idx, (data, target) in enumerate(tqdm(train_loader)):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    print(
        f"Train Epoch: {epoch} \t"
        f"Loss: {np.mean(losses):.6f} ")
for epoch in range(1, 10):
    train_notprivate(model, train_loader, optimizer, epoch, device="cpu")

def evaluate_model(test_dl, model):
    predictions, actuals = list(), list()
```

```
    for i, (inputs, targets) in enumerate(test_dl):
      # evaluate the model on the test set
      yhat = model(inputs)
      # retrieve numpy array
      yhat = yhat.detach().numpy()
      actual = targets.numpy()
      # convert to class labels
      yhat = argmax(yhat, axis=1)
      # reshape for stacking
      actual = actual.reshape((len(actual), 1))
      yhat = yhat.reshape((len(yhat), 1))
      # store
      predictions.append(yhat)
      actuals.append(actual)
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc
 # prepare the data
train_dl, test_dl = train_loader, test_loader
print(len(train_dl.dataset), len(test_dl.dataset))
acc = evaluate_model(test_dl, model)
print('Accuracy: %.3f' % acc)
DP_model = torch.nn.Sequential(torch.nn.Conv2d(1, 16, 8, 2, padding=3), torch.nn.ReLU(),
torch.nn.MaxPool2d(2, 1),
                        torch.nn.Conv2d(16, 32, 4, 2), torch.nn.ReLU(), torch.nn.MaxPool2d(2, 1),
                        torch.nn.Flatten(),
                        torch.nn.Linear(32 * 4 * 4, 32), torch.nn.ReLU(), torch.nn.Linear(32, 10))
```

```
DP_optimizer = torch.optim.SGD(DP_model.parameters(), lr=0.05)


# Step 4: Attaching a Differential Privacy Engine to the Optimizer
privacy_engine= PrivacyEngine()
N_model, N_optimizer, dataloader = privacy_engine.make_private(
    module=DP_model,
    optimizer=DP_optimizer,
    data_loader=train_loader,
    noise_multiplier=1.0,
    max_grad_norm=1.0,
    poisson_sampling=False,
)



def get_epsilon(self, delta):
  return self.accountant.get_epsilon(delta)
# Step 5: Training the private model over multiple epochs
def train(model, train_loader, optimizer, epoch, device, delta):
    model.train()
    criterion = torch.nn.CrossEntropyLoss()
    losses = []
    for _batch_idx, (data, target) in enumerate(tqdm(train_loader)):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
```

```
        optimizer.step()
        losses.append(loss.item())
    epsilon=privacy_engine.get_epsilon(delta)
    print(
        f"Train Epoch: {epoch} \t"
        f"Loss: {np.mean(losses):.6f} "
        f"(ε = {epsilon:.2f}, δ = {delta})")

for epoch in range(1, 10):
    train(N_model, dataloader, N_optimizer, epoch, device="cpu", delta=1e-5)
acc = evaluate_model(test_loader, N_model)
print('Accuracy: %.3f' % acc)
dataloader.batch_size
examples = enumerate(dataloader)
batch_idx, (example_data, example_targets) = next(examples)
fig = plt.figure()
for i in range(6):
  plt.subplot(2,3,i+1)
  plt.tight_layout()
  plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
  plt.title("Ground Truth: {}".format(example_targets[i]))
  plt.xticks([])
  plt.yticks([])
fig
print(N_model)
model_child = list(N_model.children())
import torch.nn as nn
no_of_layers=0
```

```
conv_layers=[]
for child in model_child:
  if type(child)==nn.Conv2d:
    no_of_layers+=1
    conv_layers.append(child)
  elif type(child)==nn.Sequential:
    for layer in child.children():
      if type(layer)==nn.Conv2d:
        no_of_layers+=1
        conv_layers.append(layer)
print(no_of_layers)
img = example_data
results = [conv_layers[0](img)]
for i in range(1, len(conv_layers)):
    results.append(conv_layers[i](results[-1]))
outputs = results
for num_layer in range(len(outputs)):
    plt.figure(figsize=(50, 10))
    layer_viz = outputs[num_layer][0, :, :, :]
    layer_viz = layer_viz.data
    print("Layer ",num_layer+1)
    for i, filter in enumerate(layer_viz):
        if i == 16:
            break
        plt.subplot(2, 8, i + 1)
        plt.imshow(filter, cmap='gray')
        plt.axis("off")
    plt.show()
```

```
plt.close()
```

Bibliography

[1] “5 data breaches: From embarrassing to deadly” -CNN Money, Accessed May 11, 2021. https://money.cnn.com/galleries/2010/technology/1012/gallery.5_data_breaches/index.html

[2] “Anonymized” data really isn’t—and here’s why not” - Ars Technica, May 11, 2021, https://arstechnica.com/tech-policy/2009/09/your-secrets-live-online-in-databases-of-ruin/

[3] L. Sweeney “k-anonymity: A model for protecting privacy.” - International J. of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10, No. 05, pp. 557-570 (2002).

[4] “K-Anonymity: Everything You Need to Know (2022 Guide)”-Immuta.com, Accessed May 14, 2021, https://www.immuta.com/articles/k-anonymity-everything-you-need-to-know-2021-guide/