CS2080: Applied Privacy for Data Science
Machine Learning under DP

School of Engineering & Applied Sciences
Harvard University

March 24, 2022

Discussion

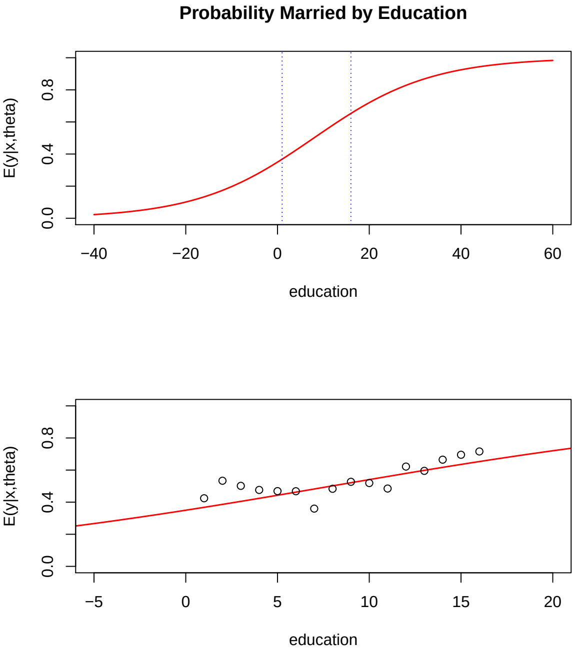
Assume we want to learn the relationship between education and income, in a sample of private data.

- Sketch out three differentially private approaches to learning this? (Make any assumptions you need but write them down.)
- Which of your methods would work if we further extended the relationship to many features/variables/covariates?
- **If time:** Does it matter if our model is predictive or inferential?
- **If time:** What would be an attack on this model if it were released without privacy-preservation?

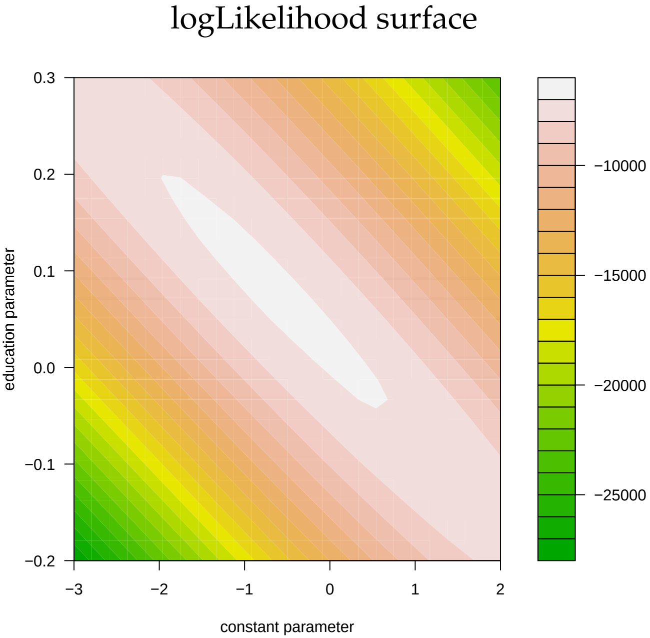
DP Optimization of Complex Models

Logit Model

$$
\begin{aligned}
logL(y|x,\theta) &= \sum_{i=1}^{N} y_i \log(\pi_i) + (1-y_i)\log(1-\pi_i), \\
\pi_i &= \frac{1}{1+e^{-\beta_0-\beta_1 x_i}}.
\end{aligned}
$$



Probability Married by Education
E(y|x,theta)
0.0
0.4
0.8
−40
−20
0
20
40
60
education
E(y|x,theta)
0.0
0.4
0.8
−5
0
5
10
15
20
education


logLikelihood surface
education parameter
constant parameter
0.3
0.2
0.1
0.0
−0.1
−0.2
−3
−2
−1
0
1
2
−10000
−15000
−20000
−25000

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

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

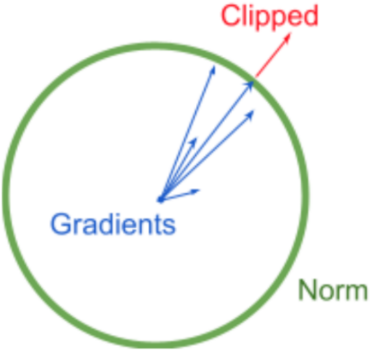
Add noise

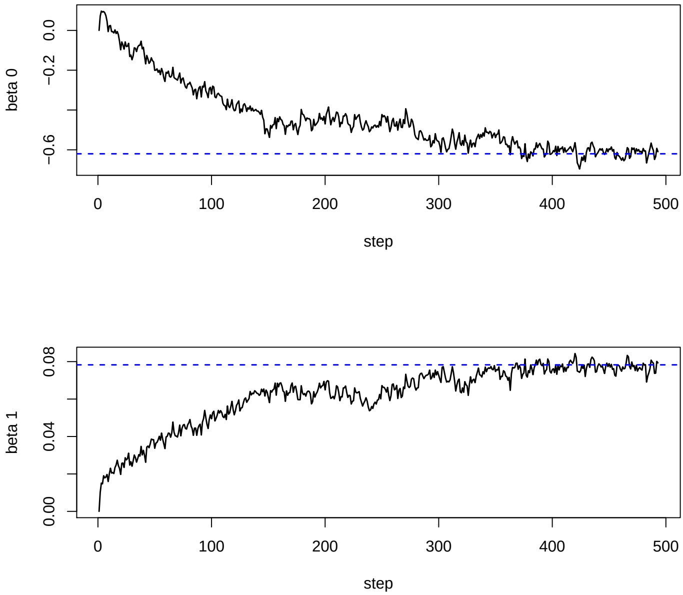
$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.


Clipped
Gradients
Norm


beta 0
0.0
−0.2
−0.6
0
100
200
300
400
500
step
beta 1
0.00
0.04
0.08
0
100
200
300
400
500
step

Parameter Tuning

Learning Rate	η_t
Clipping Norm	C
Batch Size	L

Approaches to parameter tuning:

Parameter Tuning

Learning Rate	η_t
Clipping Norm	C
Batch Size	L

Approaches to parameter tuning:

- Public data and transfer parameters
  (*Deep Learning with Differential Privacy* [Abadi *et al.* 2016])
  - Find similar styled public data, tune parameters there, transfer.

Parameter Tuning

Learning Rate	η_t
Clipping Norm	C
Batch Size	L

Approaches to parameter tuning:

- Exponential mechanism over private models
  (*Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism* [Raskhodnikova & Smith 2016])
  - Requires score function that has low sensitivity
  - Use (generalized) exponential mechanism over models

Parameter Tuning

Learning Rate	η_t
Clipping Norm	C
Batch Size	L

Approaches to parameter tuning:

- Private selection
  (*Private Selection from Private Candidates* [Liu & Talwar 2019])
  - Requires DP score function
  - Randomized stopping algorithm tunes parameters an indefinite period of time
  - however, lower expected computation and lower privacy consumption.

Broader Choices

- Instance level gradients
- Mechanisms
- Batch Sampler (Tensorflow Chunking, Opacus Uniform with replacement across batches)
- Composition
- DP definition

Opacus

Train PyTorch models with Differential Privacy

INTRODUCTION | GET STARTED | TUTORIALS

Key Features

Scalable

Vectorized per-sample gradient computation that is 10x faster than microbatching

Built on PyTorch

Supports most types of PyTorch models and can be used with minimal modification to the original neural network.

Extensible

Open source, modular API for differential privacy research. Everyone is welcome to contribute.

https://opacus.ai

Opacus for PyTorch

Write out a standard PyTorch model:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class ExampleLogisticModule(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.linear = nn.Linear(input_size, 1)

    def forward(self, x):
        x = self.linear(x)
        x = torch.sigmoid(x)
        return x[:,0]
```

Opacus for PyTorch

Write out a standard PyTorch model:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class ExampleLogisticModule(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.linear = nn.Linear(input_size, 1)

    def forward(self, x):
        x = self.linear(x)
        x = torch.sigmoid(x)
        return x[:,0]
```

Swap out the optimizer for DP:

```python
from opacus import PrivacyEngine

privacy_engine = PrivacyEngine()
model, optimizer, data_loader = privacy_engine.make_private(
    module=model,
    optimizer=optimizer,
    data_loader=data_loader,
    noise_multiplier=1.0,
    max_grad_norm=0.5,
)
```