CS168 Spring 2022, Final Project

Demystifying Differential Privacy

SUNet ID(s): avelu, preey, samar99, svaidyan
Name(s): Akash Velu, Preey Shah, Samar Khanna, Skanda Vaidyanath

By turning in this project, I agree by the Stanford honor code and declare that all of this is work of my group.

Introduction

For this final project, we have designed a mini-project for the topic Differential Privacy (DP) [3]. We investigate the meaning of ϵ and δ in (ϵ, δ)-DP with simple, intuitive experiments. We then study the compositionality property of DP algorithms and show that the composition of two ϵ-DP algorithms is a 2ϵ-DP algorithm. We then investigate how DP the typical neural network training process is and compare it to other rigorous DP approaches in terms of performance and DP-ness.

The code for this project can be found at this GitHub link

Reflection

We chose to explore this mini-project because we found the topic of Differential Privacy very interesting, as it was something we all did not know much about beforehand. Because the fundamental ideas presented in class regarding DP were thought-provoking but unintuitive, we aimed, through this mini-project, to develop questions which helped build up a basic intuition for what DP is trying to achieve, and what the different components of a DP algorithm are. Hence, we design Part 1 to closely examine the *definition* of what it mean for a randomized algorithm to be differentially private; we explore computationally through concrete examples the impact of the ϵ and δ parameters, and discuss the consequences of setting these parameters to particular values. In Part 2, we look at additional properties of DP methods, specifically compositionality, which helps build further understanding of what DP means and how it can be interpreted from an attackers perspective. In the last part, we move towards a more practical, modern example by examining DP as it relates to neural network training. Our goal through this project was to gain a better understanding of DP for ourselves, and we believe that this step-by-step construction walk-through of core DP concepts helped us and lended itself to a fun mini-project. We also aim to scaffold each question as much as possible with the relevant information regarding the topic of the question.

Part 1

We know that an algorithm is said to be (ϵ, δ)-DP if it satisfies the following:

$$\mathbb{P}[A(D_1) \in S] \leq e^{\epsilon}\mathbb{P}[A(D_2) \in S] + \delta$$

Here, D_1 and D_2 are two datasets that differ in just one record. A is a randomized algorithm that is (ϵ, δ)-DP. S is some subset of outputs in A's range.

In this part of the mini-project, we will try to gain an intuitive understanding of what ϵ and δ mean and how we can interpret them. Some of the experiments in this part are motivated by this series of blog posts [2]. The code for this is part in the file `part1.py` in the GitHub link above.

Let's start with ϵ.

The randomized coin toss experiment: Consider a population of 100 people each with an unbiased coin. All of them toss the coin and get either heads or tails; we record everyone's results in a database D_1. D_1 is the true database. Now say the first person in the database is Victoria (V) and is the target of our hypothetical attacker Ken (K). K knows the exact coin toss result of everyone else in the population except V. We would like to develop an $(\epsilon, 0)$-DP algorithm that returns the sum of heads in the population without revealing V's coin toss directly. Note that outputting the sum from D_1 directly will tell K exactly what V's coin toss is because he knows all the other coin toss results. So we need a randomized algorithm A that will give K reasonable doubt about V's result.

Consider the following $(\epsilon, 0)$-DP algorithm A: in D_1, randomly choose a proportion p rows of the dataset to alter. Keep the other rows as is. Here $0 < p < 1$. For these chosen rows, toss an unbiased coin again and record a new result for these rows. Now A just returns the sum of the new results in the dataset.

The idea is that if we have two datasets now: say the true dataset D_1 and another dataset D_2 where we alter V's result from heads to tails or vice versa. Our algorithm A should output the same output for both these datasets with high probability. This means that with the algorithm's output, K can't really figure out which dataset is the real one and hence cannot figure out what V's value is. In other words, regardless of what V's value was, the algorithm could have still outputted the same value.

Here's yet another way to think of the same problem. Since K already knows all the other coin tosses except V's and hence the sum of all the other coin tosses, we can focus only on V's coin toss. We can think of our algorithm A as just outputting the randomized result of V's toss now. In other words, we are considering a size 1 dataset with only V's toss and would like to compare how often the algorithm would reveal V's true toss outcome and how often it would reveal the opposite outcome.

(a) Say V's true outcome is a head. What is the probability that the algorithm A outputs head after randomization for V's toss alone? Write your answer in terms of p.

Solution $(1-p)+p/2 = 1-p/2$

(b) Derive an expression for ϵ in terms of p.

Solution $\ln(1-p/2) - \ln(p/2)$

(c) Let's plot a curve to see how ϵ changes with p. What trends do you notice in the plot? Do the trends make sense? Now that we have an expression for ϵ in terms of p, can we just change p to make our algorithm as DP as possible i.e. alter p such that ϵ keeps reducing? What is the issue with doing this?

Solution The plot is shown below. The issue with adding more noise is that our algorithm is trading off DP-ness for accuracy.

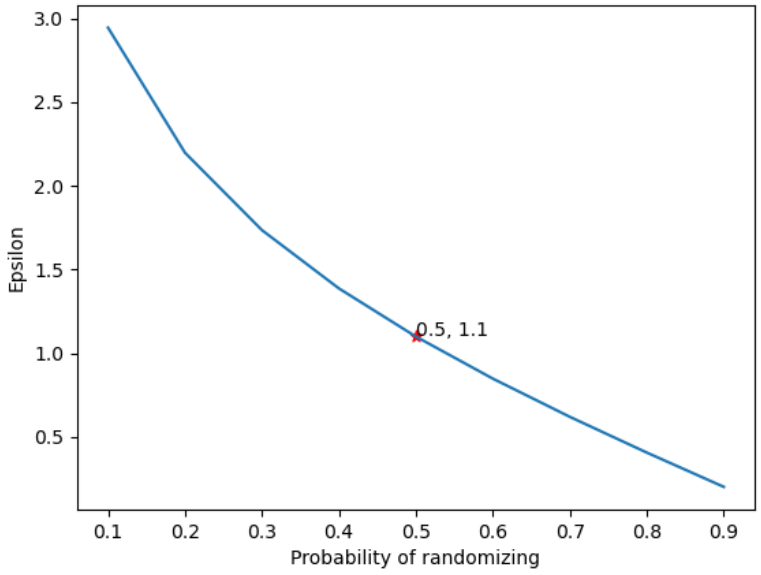

Figure 1: Part 1c

Suppose V's coin-toss result was a Heads (H), e.g. in the true database D_1, V's toss is recorded as a Heads. Let D' be a database identical to D_1, but with V's result flipped to a tails. Let D be the random variable denoting what K thinks the true database is (D can either be D_1 or D' since we assume K knows everything about D_1 other than V's result).

We want to predict how K will react after observing the outcome of A. To do this, you first determine K's a-priori belief distribution about V's coin toss result, $\mathbb{P}[D = D_1]$. Given this prior distribution, we can compute a *posterior* distribution of K's updated belief about

V's coin toss outcome after observe the output of A: $\mathbb{P}[D = D_1|A(D) = N]$. We can show that this posterior is bounded in the following manner:

$$\frac{\mathbb{P}[D = D_1]}{e^\epsilon + (1 - e^\epsilon)\mathbb{P}[D = D_1]} \leq \mathbb{P}[D = D_1|A(D) = N] \leq \frac{e^\epsilon \mathbb{P}[D = D_1]}{1 + (e^\epsilon - 1)\mathbb{P}[D = D_1]}$$

In the following part, we will study the impact of the prior $\mathbb{P}[D = D_1]$.

(d) Let's examine what happens when p and the prior $\mathbb{P}[D = D_1]$ vary.
For $p = [0.1, 0.2, 0.5, 0.7, 0.9]$, plot curves of the upper and lower bounds given above as a function of $\mathbb{P}[D = D_1]$. Plot these curves by computing the posteriors for $\mathbb{P}[D = D_1] = [0.0, 0.1, ..., 1.0]$ and plotting a line curve of these values. You should plot 5 different curves, one for each of the values of p.

What trends do you notice with respect to p and $\mathbb{P}[D = D_1]$?

Solution The plot is shown below. The curves with smaller p are outside showing that the lower ϵ-DP algorithms gives more information to the attacker than the higher ϵ ones.

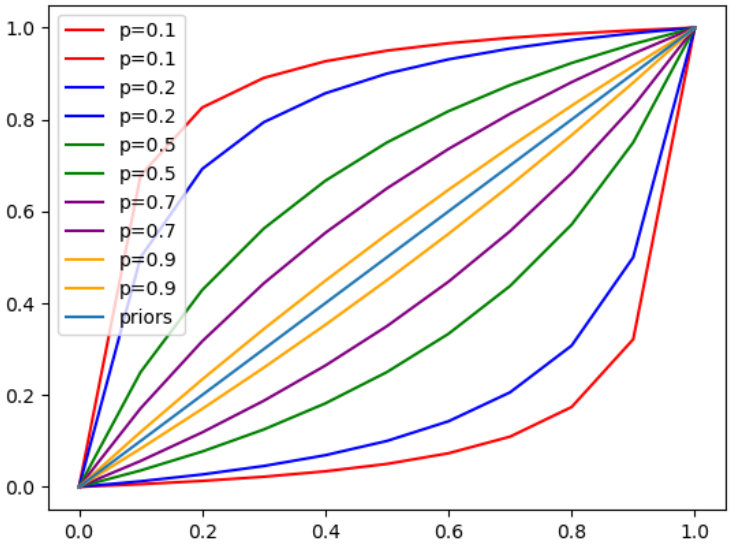

Figure 2: Part 1d

(e) Set $p = 0.5$. Plot a curve of the upper bound and lower bound (given by the equations above) as a function of $\mathbb{P}[D = D_1]$. Again, compute the posterior for $\mathbb{P}[D = D_1] = [0, 0.1, 0.2, 0.3, ...1.0]$ and plot a line curve over the outputs.

Bayes' Rule tells us that K's updated belief after observing the output of A will be $\mathbb{P}[D = D_1 | A(D) = N] = \frac{\mathbb{P}[D=D_1] \cdot \mathbb{P}[A(D)=N | D=D_1]}{\mathbb{P}[A(D)=N]}$.

For these values of the prior, run the randomized algorithm A on D_1 and compute the posterior above. Do the same after running the algorithm on D'.

Hint: you should **estimate** $\mathbb{P}[A(D) = N]$ via by running A many times and determining the # of times the algorithm outputs N. Plot these values as a scatter plot in the same figure as the upper and lower bound curves.

Do the posterior values computed above fall within the upper and lower bounds?

Solution Yes, the computed posterior values fall within the theoretical upper and lower bounds, as seen in the chart

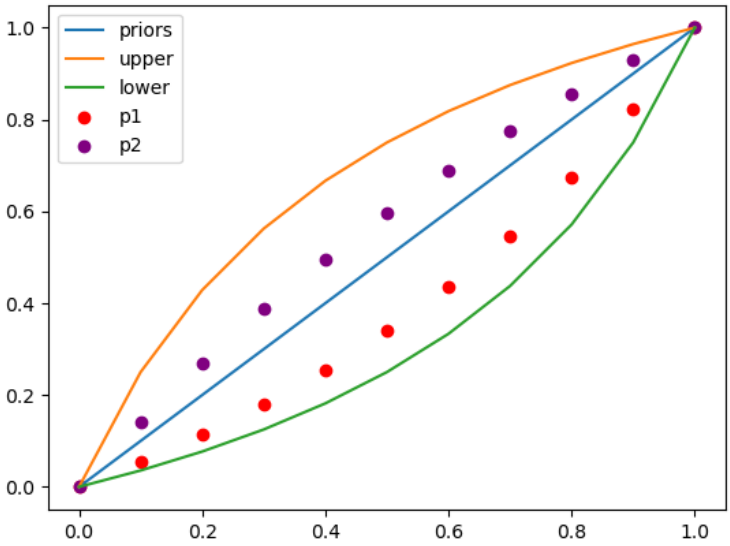

Figure 3: Part 1e

Now that we have a good idea of how ϵ works, lets try to gain some intuition on δ.

Consider an example where we have a dataset D_1 of favourite sports collected from the Stanford community. Say this list contains the sports Tennis, Volleyball, Basketball and Baseball in different frequencies. Say we have a fake dataset D_2 that contains all the sports listed in D_1 (perhaps in different frequencies) but also a new sport Football that occurs exactly once. Say we want an algorithm A to output a histogram for each dataset for the frequency count of each sport in the dataset. An attacker should not be able to see the output of A and determine which dataset was used to generate the histogram.

How does this map to a useful real-world example? Say the attacker wants to find Samar's degree programme at Stanford. The attacker know that D_1 is collected from the Engineering school and D_2 from the Business school. If the attacker further knows that Samar likes Football, if A outputs the frequency histogram without a DP approach, the attacker can directly infer Samar's degree programme using this information.

Consider the following (ϵ, δ)-DP algorithm A. Add Laplacian noise with parameter $1/\epsilon$ to each count and then remove all counts that are lower than some threshold T. The Laplacian noise term leads to the ϵ part of the DP algorithm and the thresholding gives the δ part.

Lets focus on the threshold T part of the algorithm. Say $T = 5$ – we'd like to prune off the Football entry in D_2 to make sure there are no distinguishing events. However, when we add Laplacian noise, there is a small chance that we add enough noise to make the count of Football go from 1 (in the true dataset) to 5 or above. This "small chance" is exactly δ and with this probability, we will retain Football in our output, thereby directly revealing the dataset used. Luckily, when we use Laplacian noise of parameter $1/\epsilon$ where $\epsilon = \ln 3$, this happens with probability roughly 0.6% (to see how, check out the Laplace distribution's CDF).

(f) Let's simulate this algorithm by running A on D_2 10000 times and see how many times it reveals the Football sport. How many times does the algorithm "fail"? Is this value close to $\delta = 0.6\%$?

Solution The output is shown below. It can vary a bit with variance in the simulation.

```
Likelihood of seeing Football in D (approx delta): 0.005
(base) ➜  differential-privacy git:(main) ✗ python part1.py
Likelihood of seeing Football in D (approx delta): 0.007
(base) ➜  differential-privacy git:(main) ✗ python part1.py
Likelihood of seeing Football in D (approx delta): 0.005
(base) ➜  differential-privacy git:(main) ✗ python part1.py
Likelihood of seeing Football in D (approx delta): 0.004
(base) ➜  differential-privacy git:(main) ✗ python part1.py
Likelihood of seeing Football in D (approx delta): 0.006
```

Figure 4: Part 1f

(g) Once again, it seems like we can keep increasing T to get lower δ values which is desirable. However, is this prudent to do? Why or why not?

Solution No, increasing threshold will tend to prune off more and more sports that don't have a small frequency.

Part 2

Another interesting property of DP algorithms is that they are compositional. In other words, if we have an ϵ-DP algorithm A and an ϵ-DP algorithm B. Then, an algorithm that publishes the results of both A and B is 2ϵ-DP. This is easy to show when the two algorithms are independent.

(a) Prove the compositionality result stated above.

Hint: Let C be the algorithm that publishes the results of A and B as $O = (O_A, O_B)$, D_1 be the true dataset, and D_2 be a dataset with one record altered from D_1.

Solution

$$\begin{aligned}
\mathbb{P}[C(D) = (O_A, O_B)] &= \mathbb{P}[A(D_1) = O_A, B(D_1) = O_B] \\
&= \mathbb{P}[A(D_1) = O_A] \cdot \mathbb{P}[B(D_1) = O_B] \\
&\leq e^{\epsilon}\mathbb{P}[A(D_2) = O_A] \cdot e^{\epsilon}\mathbb{P}[B(D_2) = O_B] \\
&= e^{2\epsilon}\mathbb{P}[A(D_2) = O_A] \cdot \mathbb{P}[B(D_2) = O_B] \\
&= e^{2\epsilon}\mathbb{P}[C(D_2) = (O_A, O_B)] \\
&\implies \text{C is } 2\epsilon - \text{DP.}
\end{aligned}$$

(b) Lets try to simulate the above to ensure that algorithm C is 2ϵ-DP. To do this, we go back to the randomized coin toss experiment from part 1 and set A and B to be ϵ-DP algorithms with $p = 0.5$. We will show that C is a 2ϵ-DP algorithm using a similar simulation procedure as we did in part 1. First, we get the outputs of A and B on D_1. Next, we find the probability of this concatenated output under C (which is just a concatenation of outputs of A and B) under datasets D_1 and D_2. We estimate ϵ of C using these probabilities and simulate this 100 times and take the maximum of all ϵs we obtain to get a final estimate. We take the max because we want to find the "worst possible" ϵ in a sense. You should find that this value is lower than (or slightly higher) than the theoretical upper bound of 2ϵ. You may find that there is some variance because of the number of trials and our approximate method of finding ϵ.

The code for this is given in the file `part2.py` in the GitHub link above.

Solution The output is given below. It may vary a little with noise.

Figure 5: Part 2b

Part 3

We've seen some basic (ϵ, δ)-DP algorithms for some toy problems. Lets move to a slightly more realistic problem that we might care about. Given a train dataset $\mathcal{D}_{\text{train}}$ of training points, we want to train a model to minimize mean test error on an unseen test set $\mathcal{D}_{\text{test}}$. Lets say this is a regression problem for simplicity, so we want to minimize MSE. Let the model be a simple two-layered neural network.

Consider the following randomized algorithm A:

- Initialize the weights of the model randomly using Xavier Uniform Initialization [4]. This provides randomness in the algorithm
- Perform batch gradient descent on the dataset for 5 epochs to optimize the model. The order of batches may also add randomness to the algorithm
- Test the algorithm on the test set and return the mean test loss rounded off to 2 decimal places

The algorithm described above is a typical neural network training procedure that does not take any DP-ness into account. We would like to find the ϵ of this process assuming A is $(\epsilon, 0)$-DP.

(a) The code in `part3.py` in the GitHub link runs this experiment for you and outputs the estimated ϵ using a procedure similar to part 2. What is the value of ϵ you observe?

Solution The output is shown below. It may vary across runs because of noise in the simulation

We found that the experiment above was not very consistent and had a large variance in the estimated ϵ values across runs. This is expected as this method of estimating ϵ is very approximate and is done using several approximations in simulation.

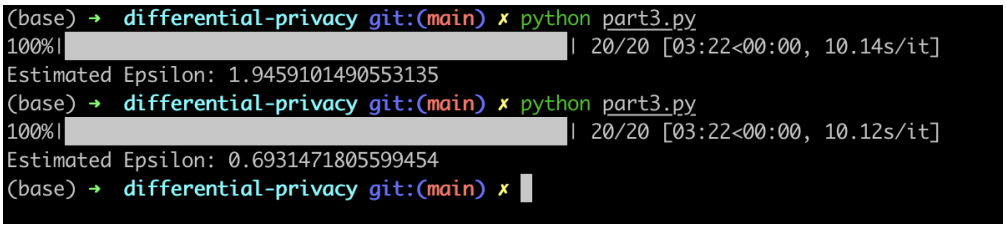

Figure 6: Part 3a

There are other DP algorithms to train neural networks [1, 5] that are more rigorous and can theoretically show that the resulting algorithm is (ϵ, δ)-DP instead of trying to find the ϵ value post-facto.

We tried running DP-SGD [1] on a similar setup to compare the ϵ from that algorithm with the theoretical guarantees and the ϵ from the naive neural network training procedure. However, we couldn't get DP-SGD to train on the data properly. Other recent works [5] have also noted that DP algorithms often tendn to work much worse than their non-private counterparts. We have added the code to the `part3.py` file for future investigation.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2016. 9

[2] Damien Desfontaines. A friendly, non-technical introduction to differential privacy (link). 2021. 2

[3] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 1

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *AISTATS*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. 8

[5] Alexey Kurakin, Shuang Song, Steve Chien, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. Toward training at imagenet scale with differential privacy, 2022. 9