Code guide

July 31, 2020

This document describes the code that implements the differentiallly private algorithms in Differentially Private Survey Research. This is an on-going project and so some algorithms described in the paper are not yet available in the software.

1. Laplace Mechanism (via noise on the local device)

In this section we describe the code in `local_laplace.R`. At a high level, the code takes a set of n survey responses and produces a differentially private histogram by adding specially calibrated noise locally to each of the n devices (e.g., a cell phone).[1]

Raw data inputs

For simplicity we'll use an example where there are two survey questions (V1 and V2) each with a binary response option (0/1). But everything generalises to cases with an arbitrary number of questions with an arbitrary number of response options.

Start with a respondent answer vector:

```
respondent_vector <- c(V1 = 1,V2 = 0)
respondent_vector
```

```
## V1 V2
##  1  0
```

We then need a 'key', which is a matrix of all possible answer combinations. Here there are 4 possible answers:

```
key <-expand.grid(V1 = c(0, 1), V2 = c(0, 1))
key
```

```
##   V1 V2
## 1  0  0
## 2  1  0
## 3  0  1
## 4  1  1
```

[1] A worked example is found in this repository under `examples/local_laplace_example.R`.

One hot encoding

The first step is to represent the respondent vector as a one-hot encoding, which here is a vector of length 4 (the number of possible answers combinations, K) with a 1 in the position of the row in `key` that matches the respondent's answer vector. The function that does this is `oneHot()`. It takes the `respondent_vector` and `key` as its two arguments. For example, in this case the 2nd row in `key` is the match:

```
one_hot <- oneHot(respondent_vector, key)
one_hot
```

```
## [1] 0 1 0 0
```

Local Differential Privacy[2]

The next step is to take this *private* one hot encoding, `one_hot`, and make it differentially private by adding specially calibrated noise.

The noise we add is Polya$(1/n, \alpha)$ - Polya$(1/n, \alpha)$. The function `polya()` takes one draw from a Polya distribution where the arguments are n and *epsilon* (the privacy paramater).[3] We draw from this Polya distribution in two steps:

1. $\lambda_i \sim \text{Gamma}\left(\frac{1}{n}, \frac{\alpha}{1-\alpha}\right)$
2. $Z_i \sim \text{Pois}(\lambda_i)$

where $Z_i \sim \text{Polya}(1/n, \alpha)$.

It is important to recognise that we do not simply use R's in-built functions for taking draws from the Gamma and Poisson distributions (`rpois` and `rgamma`) since these functions rely on a form random number generation deemed to be insufficient for differential privacy. The problem, in short, is that they rely on deterministic processes that are merely made to look random, rendering them vulnerable to privacy attacks. Thus we instead use crytographically secure psuedo-random number generataion (CSPRNG), drawing heavily from the OpenSSL library. There are three relevant functions used inside of `polya()`:

- `customUniform()` uses the openSSL library to take crypotgraphically secure draws from Unif$[0, 1)$
- `customGamma()` takes secure draws from a gamma distribution[4]
- `customPoisson()` takes secure draws from a poisson distribution[5]

[2]We are extremely grateful to Christian Covington for sharing his wise advise and code in relation to this section.

[3]We then calculate α as a deterministic function of *epsilon*.

[4]The function uses the GS algorithm from Ahrens and Dieter found on p.425 of http://www.eirene.de/Devroye.pdf

[5]This implements the 'inversion of sequential search algorithm' found on p.505 of http://www.eirene.de/Devroye.pdf

Now we have a set of functions to draw securely from the Polya distribution. The function `deviceNoise()` then takes K differences between draws of Polya noise from `polya()`, one for each element of the vector `one_hot`. So for example:

```
set.seed(123)
epsilon <- 0.5
n <- 100
K <- nrow(key)
noise <- deviceNoise(K, n, epsilon)
noise
```

```
## [1] 0 0 0 0
```

The function `deviceDP()` is a wrapper function for these steps. It takes as its arguments a private one hot encoding `one_hot`, `n` (the number of respondents) and `epsilon` (the privacy paramater). The function adds `noise` to `one_hot` and returns a sparse representation of this differentially private one hot encoding. Since most entires will be zero, we return the data in a sparse form as a matrix with two columns and an arbitrary number of rows, with the position of non-zeroes in the first column and the associated values in the second column. So below we see that there is a non-zero in the 2nd position of the DP one hot encoding and the value was 1:

```
set.seed(456)
respondent_data <- deviceDP(one_hot, n, epsilon)
respondent_data
```

```
##      [,1] [,2]
## [1,]    2    1
```

Server aggregation

Once we have this differentially private one hot encoding, we can send `respondent_data` to the server. The final piece of code aggregates a list of n outputs from `deviceDP()`. The function that does this step is `dpHistogram()`. We break this function down in steps below.

To demonstrate the functionality, we first need to simulate a list of DP encodings. Here we just repeatedly add random noise to the same `respondent_vector`, bur in practice we would have `n` different vectors.

```
set.seed(789)
one_hot_list <- replicate(n, deviceDP(one_hot, n, epsilon))
```

`dpHistogram()` takes a list of this nature as its first argument, and `key` as the second. The first operation in the function is to convert this list into a data frame, and to then take grouped sum (grouped by the first column which is the position in `key`). This

gives a vector of noisy counts for the number of respondents whose answers were in each row of `key`. Finally we match the vector of noisy counts back to `key`. This produces a coherent representation of the individual level data (with differentially private noise of course).

```
dpHistogram(one_hot_list, key)
```

```
##   V1 V2 n_response
## 1  0  0          4
## 2  1  0        100
## 3  0  1          1
## 4  1  1         -2
```