Rappor on Groups *Akash Jain*

Background

Consider some universe of websites $U = \{w_1, w_2, \ldots, w_{100}\}$ and N clients.

For clarity, we'll use the following example: $U = \{$"Google", "Facebook", "Snapchat", "Tinder", "Instagram", "CNN", "NYTimes", "Economist", "Piazza", "Bloomberg", "Pinterest", "FT", "Guardian"$\}$, where $|U| = 13$.

Each client i differentially-privately reports some subset S_i of U to the server. These are the websites that client i visits. The server aggregates these reports from all the clientsand creates an estimate of the fraction of clients that visit each website in U. Thus, the server will produce a vector of frequencies $F = \{f_1, f_2, \ldots, f_{100}\}$. f_k is an estimator of the quantity $\frac{\sum_{i=0}^{N} \mathbb{1}_{w_k \in S_i}}{N}$

Motive

The server wishes to be able to answer queries such as "What is the probability that a Snapchat user also uses Tinder?" or "Are consumers of online news more likely to visit Instagram or Facebook?". More generally, the server wants to know (in aggregate for all clients) $\mathbb{P}(w_7 | w_{14}, w_{85}, \ldots, w_{93})$.

However, naively reporting this information using RAPPOR can run into some difficulties. There are $\binom{100}{5} \approx 7.5 \cdot 10^7$ possible permutations of 5 websites so accurately trying to recover which of these quintuplets is the most common list of top 5 websites for a given client is very difficult because of the noise added to client reports by the RAPPOR algorithm. To solve this, I propose the following algorithm:

Algorithm

The broad outline of the algorithm is ...

Round 1

At the beginning, clients simply report to the server the websites in U that s/he visits.

$$S_i = \{w_7, w_{15}, w_{19}, \ldots, w_{88}\}$$

The server aggregates all the S_is from clients to determine which websites are commonly visited. From these, the server can pick which website s/he is interested in to learn more

about, e.g. w_{81}. The server then returns w_{81} to all of the clients, revealing no more information about the choices of any individual client.

Round 2

If the client hasn't visited w_{81}, s/he stops sending any more reports (or sends dummy bits). Otherwise, the client sends the websites s/he visited as before.

The server again can aggregate reports from all the clients to create a vector $F = \{f_1, f_2, \ldots, f_{100}\}$. Here, each f_i represents $\mathbb{P}(w_i|w_{81})$. The server can now pick the next website it is interested in and send it back, along with w_{81} from Round 1. Say the server sends back (w_{81}, w_{64}).

Round 3

As before, if the client hasn't visited *both* w_{81} *and* w_{64}, s/he reports either nothing or dummy bits. Otherwise, the client sends the websites s/he visited.

As before, the server aggregates reports from all the clients to create a vector $F = \{f_1, f_2, \ldots, f_{100}\}$. Here, each f_i represents $\mathbb{P}(w_i|w_{81}, w_{64})$. The server can now pick the next website it is interested in and send it back, along with w_{81} and w_{64} from earlier rounds. Say the server sends back (w_{81}, w_{64}, w_{49}).

Round q

The server can continue this process as long as desired until it has found its desired result.

The algorithm can be used both in an exploratory sense as well as an analytical sense. If you know which websites you desire to predict, you can simply find the probability by sending the appropriate websites back from the server.

If you want to explore the data to find interesting trends in the data, you can also set the server to return the most common websites after each round.

Finally, you can employ a mix of both messages (e.g. pick the most popular news website for the first 5 rounds).