HW 1: Reidentification, Reconstruction and Membership Attacks

CS 208 Applied Privacy for Data Science, Spring 2019

Version 1.0: Advance preview of HW1, two more problems to follow after Monday's lecture.

Instructions: Submit a single PDF file containing your solutions, plots, analyses, and documented code. Also include a link to a public repository with your code (such as GitHub/GitLab). Make sure to list all collaborators and references.

1. **Reidentification Attack**

   In the GitHub repo[1], you will find the Public Use Micro Sample (PUMS) dataset from the 2000 US Census `FultonPUMS5.csv`. This is a sample from the "Long Form" from Georgia residents, which contained many more questions than the regular questionnaire, and was randomly assigned to some individuals during the decennial Census. (It has since been replaced by a continuously collected survey known as the *American Community Survey*.)

   Also in that folder is the codebook file for the PUMS dataset that lists the variables available in the release. Note this is the 5% sample which means that five percent of records are randomly sampled and released.

   In the style of Latanya Sweeney's record linkage reidentification attack[2], propose a reidentification attack on the PUMS dataset by identifying demographic variables that, if known from another auxiliary source, could uniquely identify individuals. Note that while Sweeney used zipcodes as the geographic indicator, individuals in this Census release are identified by Public Use Microdata Areas (PUMAs) which are Census constructed geographic areas that contain at least 100,000 individuals. State the variables you would use, and provide an approximate back-of-the-envelope calculation of the number of individuals who would be unique in that combinations of variables in a PUMA region.

2. **Reconstruction Attack**

   Among the variables in the 2000 PUMS dataset above is `USCITIZEN`, which asks the resident about their US Citizenship status. This is a sensitive piece of information, and including this question on the regular Census questionnaire has been a topic of recent controversy.[3] This PUMS dataset is public, but makes a good stand-in for a database that might be secured behind a query interface. We've provided a sample of size $n = 1000$.

   In this problem, you will run experiments to evaluate the performance of the reconstruction attack on determining individuals' citizenship status. Treat the following variables in the dataset

[1] https://github.com/privacytoolsproject/cs208/tree/master/data

[2] https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1748-720X.1997.tb01885.x

[3] See e.g. https://www.nytimes.com/2019/01/15/us/census-citizenship-question.html

as public (so as an attacker you know them for all of the individuals in the dataset):

$$\texttt{PUB} = (\texttt{SEX}, \texttt{AGE}, \texttt{EDUC}, \texttt{AGE}, \texttt{MARRIED}, \texttt{DIVORCED}, \texttt{LATINO}, \texttt{BLACK}, \texttt{ASIAN}, \texttt{CHILDREN}, \texttt{EMPLOYED}, \texttt{MILITARYSERVICE}, \texttt{DISABILITY}, \texttt{ENGLISHABILITY}).$$

Each query in your attack should specify a boolean predicate $p(\texttt{PUB}) \in \{0,1\}$ on the public variables (e.g. $p(\texttt{AGE/EDUC} > 4 \;\&\&\; \texttt{SEX} == 0)$), and receive as an answer an approximation to the value:

$$\sum_{i:p(\texttt{PUB}_i)=1} \texttt{USCITIZEN}_i,$$

where i ranges over the $n = 1000$ individuals in the PUMS dataset sample that we have provided.

Your attack should make $10n$ queries, where each query corresponds to a different predicate p_j, $j = 1, \ldots, 10n$.[4] Using the description of these predicates, the public data $\texttt{PUB}_1, \ldots, \texttt{PUB}_n$, and the noisy answers to the queries, you should try to reconstruct the $\texttt{USCITIZEN}_i$ bits for as many users as possible.

You will run experiments on how your attack performs against the following defenses:

(a) Rounding: round each result to the nearest multiple of R for a parameter R

(b) Noise addition: add Gaussian noise of mean zero and variance σ^2, for a parameter σ, independently for each query.

(c) Subsampling: randomly subsample a set T consisting of t out of the n rows, for a paremeter t, and calculate the answer using only the rows in T (scaling up by a factor of n/t).

Varying parameters R, σ, and T as integers from 0 to n, produce plots showing and comparing the trade-off between the accuracy of the statistics (measured by root-mean-squared-error between answers and exact values) and the average fraction of values $\texttt{USCITIZEN}_i$ that are successfully reconstructed. For each parameter setting, run 10 experiments with fresh randomness and plot the average data points.

Make sure to identify the regime where your attack transitions from near-perfect reconstruction (fraction close to 1) to near-unsuccessful reconstruction (fraction close to 1/2). Add additional data points so that your graph is detailed around that transition point.

Note that you will be coding both the release mechanisms for each defense as well as the attack. The GitHub repo contains the code from the regression-based reconstruction attack from Monday's class[5]. (Be sure to pull the most recent copy.) You can directly expand from this code if you are working in R, or use it as a template if you are working in Python.

BONUS: The above attack requires knowledge of all of the $\texttt{PUB}_i$'s. Here we will sketch a version of the attack that only requires knowledge of a single $\texttt{PUB}_i$ and reconstructs $\texttt{USCITIZEN}_i$ for that particular individual. For extra credit, fill in the details and implement the attack and measure its performance.

[4] Suggestion: To create predicates that specify "random" subsets, you'll want to randomly hash a long vector of integer values $v = (v_1, \ldots, v_d)$ containing each individual's public attributes into a binary value. A good way to do this is to fix a large prime P, choose random numbers $r_1, \ldots, r_d \in \{0, \ldots, P-1\}$, and define $p(v) = ((\sum_d r_d v_d) \bmod P) \bmod 2$.

[5] At https://github.com/privacytoolsproject/cs208/blob/master/examples/wk1_attacks/ see `regressionAttack.r` and `regressionAttackOverQuerySize.r`

Above, we suggested using a random hash function of the form $p(v) = ((\sum_k r_k v_k) \bmod P) \bmod 2$ to select subsets of the dataset. Instead, consider taking P to be a prime of magnitude roughly $10n$, choosing $r_1, \ldots, r_k$ once and for all, and defining the hash function $h(v) = \sum_k r_k v_k \bmod P$. Since P is significantly larger than n, there will be few collisions of the $\texttt{PUB}_i$'s under the hash function h. Now for each query p_j, pick a random number $s_j \in \{0, \ldots, P-1\}$, and define $p_j(v) = ((s_j \cdot h(v)) \bmod P) \bmod 2$. Now you can do your linear regression with P variables, one for each possible value of $h(\texttt{PUB})$, since h is not changing across the queries. To attack a particular individual i, we look at the result of the regression for the variable associated with $h(\texttt{PUB}_i)$.