Privacy analysis

P.S Sommerfelt[1], R. Syed[1], R. Akbari[1]
T. Seierstad [2]
[1] Department of Mathematics, [2] Department of Informatics

November 5, 2021

Introduction

An important task in Data Science is privacy analysis. A good data scientist have to protect sensitive data. There are different methods that deals with this. In this section, we will introduce two different types of differential privacy techniques such as Randomized Response and Exponential Mechanism. The topics we will concentrate on is how can we make sure that the existence of a specific database does not raise any privacy concerns, and if such database was hidden but the analysis is public, how will that affect privacy?

For the rest of the report, we will mainly concentrate on the treatment data set. We will later construct a new policy for treatment decisions.

The randomized response

We can protect the treatment data by using the randomized response. For treatment data with n samples $x_1...x_n$, we transform these into $a_1...a_n$ by following local privacy model:

```
for each x_i:
    flip a coin
    if coin == head:
        a_i = x_i
    else:
        flip a coin
        if coin == head:
            a_i = x_i
        else:
            a_i != x_i
```

In other words, the model is about flipping a coin. If the flip results in head then we send in the true sample, and if it results in tail the model will flip a coin again. This time if it is a head we will annotate $a_i = x_i$ and $a_i \neq x_i$ otherwise. In this way, we make sure that the differential privacy criterion is satisfied. This means that we do not risk leaking private information about the individuals by releasing an obtained privacy and model which is trained

on a training data from a randomized response mechanism. Because the responses are totally random and the rate of true data is

$$q = 2E_p - \frac{1}{2}$$

where E_p is the expectation of rate of positive responses [1]. This leads to 0-Differential privacy [2], which will result in better privacy but less accurate response [3].

The exponential mechanism

The exponential mechanism (EM) is one of the most general privacy differential mechanisms. The ME mechanism selects and outputs an element r from a range of elements $\mathfrak{R}$ with probability proportional to

$$\exp\left(\frac{\epsilon u(x,v)}{L(u)}\right)$$

[4] where $u(x,v)$ is a utility function which maps subset of the data into utility scores and the $L(u)$ is the sensitivity of a query, i.e the maximum change between subset x and y. The best part of EM is that the privacy loss is approximately [4]

$$\ln\left(\frac{\exp\left(\frac{\epsilon u(x,v)}{L(u)}\right)}{\exp\left(\frac{\epsilon u(y,v)}{L(u)}\right)}\right) \leq \epsilon$$

In other words, the loss of privacy is less or equal to the ϵ and it preserves the ϵ-DP. This mechanism is much better than the others since it releases less information about individuals, and more about the elements with most noise. The decisions of what treatment to give to individuals can be assumed to be publicly available, and we ensure that none of individual information is leaked by using the EM.

Implementation of the exponential mechanism

The code below shows the implementation of the of the exponential mechanism. A running example of this code is available at GitHub .

```
class PrivacyPolicy:
    def __init__(self, features, actions, outcome):
        self.features = features
        self.actions = actions
        self.outcome = outcome
    def exponential_mechanism(self, epsilon):
        """
        Given a set of actions and a utility function, this
            function returns the noisy 'best' action.
        Since our utility function at the moment does not
           depend on actions, only outcome, the results are
            expected to be random.
        """
        best_actions = []
        for i in range(self.features.shape[0]):
            utility = np.array([self.get_utility(self.
                features.iloc[i,:], action, self.outcome.
                iloc[i,:]) for action in self.actions.iloc[i
                ,:]])
            policy_probs = np.exp(epsilon*utility/2*self.
                sensitivity)
            policy_probs = policy_probs/np.linalg.norm(
                policy_probs, ord=1)
            best_actions.append(np.random.choice(self.
                actions.columns, 1, p=policy_probs.ravel())
                [0])
        return best_actions
    def get_utility(self, features, action, outcome):
        utility = 0
        utility -= 0.2 * sum(outcome[['Covid-Positive']])
        utility -= 0.1 * sum(outcome[['Taste']])
        utility -= 0.1 * sum(outcome[['Fever']])
        utility -= 0.1 * sum(outcome[['Headache']])
        utility -= 0.5 * sum(outcome[['Pneumonia']])
        utility -= 0.2 * sum(outcome[['Stomach']])
        utility -= 0.5 * sum(outcome[['Myocarditis']])
        utility -= 1.0 * sum(outcome[['Blood-Clots']])
        utility -= 100.0 * sum(outcome[['Death']])
        self.sensitivity = 100
        return utility
```

Utility function

We first define some preferences A and B, and prefer A to B if $U(A) > U(B)$. Our preference should not be cyclical, e.g $A \sim B \sim C \sim A$. The set of all preferences defined given a theme, is called reward (r). The properties that the elements in the set should have is:

- For every element A and B in the set of rewards either you have A preferred to B, B preferred to A or A indifferent to B.
- If A is preferred to B and B is preferred to C then A is preferred to C.

The utility depends on some set of actions when applied to x (observations) gives a real output. The utility $U(a, y)$, a depends on x. Since the action is finite and the outcome is finite, then utility is a discrete function. The action set is defined as follows.

$$A = \{a_0, a_1, a_2, a_3\}$$

where a's are

- a_0 is the action of not treating an individual with any treatment
- a_1 is the action of treating an individual with treatment 1
- a_2 is the action of treating an individual with treatment 2
- a_3 is the action of treating an individual with treatment 2 and treatment1

The reward for each person is defined as a table. The utility becomes

$$\sum_i r_i$$

where r_i is the reward table. For each row we choose the action, given features and outcome, that maximizes the reward. We sum all the rewards for all observations to get the utility. The reward will be different depending on the state of the individual in the row.

Every symptom has a negative contribution to the reward depending on the severity of symptom. Choosing not to treat a patient with lot of symptoms will have a large negative reward. In other words, choosing to treat this patient will be the preferred action. In contrast, a patient with close to no symptoms will not benefit from a treatment and the preferred action should

be not to treat, i.e a_0. We assume that the policy is not restricted, i.e there is a treatment for everyone who needs it. If the policy was restricted, we would have to choose which patients would need the treatment the most. For example, consider the following patient

Age	60
No-Taste/Smell	yes
Fever	yes
Headache	yes
Pneumonia	yes
Stomach	yes
Myocarditis	yes
Blood-Clots	yes
Asthma	yes
Obesity	yes
Smoking	yes
Diabetes	yes
Heart disease	yes
Hypertension	yes

For the example above the policy could be something like

$$\pi(a|x) = \{a_0 : 0.0, a_1 : 0.2, a_2 : 0.2a_3 : 0.6\}$$

Sensitivity of the utility function

In order to estimate the amount of loss in utility as we change the privacy guarantee, we compute the sensitivity of the utility. It is defined as

$$L(u) = \sup_{xNx'} |u(a, x) - u(a, x')|$$

Bibliografi

[1] Chapter 3.4 in Machine learning in science and society, (https://github.com/olethrosdc/ml-society-science/blob/master/notes.pdf).

[2] Machine learning in science and society - Remark 3.4.2, (https://github.com/olethrosdc/ml-society-science/blob/master/notes.pdf).

[3] Shaistha Fathima - Sep 15h 2020, (https://medium.com/@shaistha24/differential-privacy-definition-bbd638106242).

[4] C. Dwork, A. Roth - The Algorithmic Foundations of Differential Privacy - chapter 3.4,
(https://www.cis.upenn.edu/ aaroth/Papers/privacybook.pdf).