CS591S1 Homework 1: Attacks and DP Basics

Jiawen Liu
Collaborators: none.

1 Problem 1

1. The only input to the attacker is a (the vector of noisy counters).

   - **Algorithm Description**

Algorithm 1 Reconstruction Attack without Extra Information

Require: The observed results a from query.
Initialize $c = a$, as the reconstructed counter .
for $i \in [a.length]$ **do**.
  let $s = a[i] - c[i-1]$ be the possible j_{th} item in data set.
  If $s > 1$ **do**. $c[i] = a[i] - 1$ {the correct counter at position i must be a[i] - 1}
  Elif $s < 0$ **do**.
    let $j = i$
    While $j - 1 > 0 \wedge c[j] < c[j-1]$ **do**. $j-- = 1$ $c[j]-- = 1$
{the correct counter at position i must be a[i] and decrease the previous by 1 until their difference ≥ 0}
return $s_i = c_i - c_{i-1}$ **for** $i \in [a.length]$.

   - **Experimental Results Plotting** Figure 1
   - **Results Discussion**
     The average accuracy is 0.7775 from the Alg. 4, which is higher than 0.75.
     I also got different results form some attempts before I achieved this result:
     attempt 1. I first tried the naive method in Alg. 2, which only achieved standard 0.75 accuracy on average.

Algorithm 2 Naive Reconstruction Attack without Extra Information

Require: The observed results a from query.
Initialize vector s, $s[i] = a[i] - a[i-1]$, as the reconstructed dataset .
for $i \in [a.length]$ **do**.
  If $s > 1$ **do**. $s[i] = 1$
  Elif $s < 0$ **do**. $s[i] = 0$
return s.

     Attempt 2. Then I tried the minimizing error method from class in Alg. 3, which achieved a better accuracy but is inefficient.
     Attempt 3, Alg. 4. Then I go back to the Alg. 2 and made some improvements in it.
     – Instead of recovering the data base directly, I firstly recovered the counter.

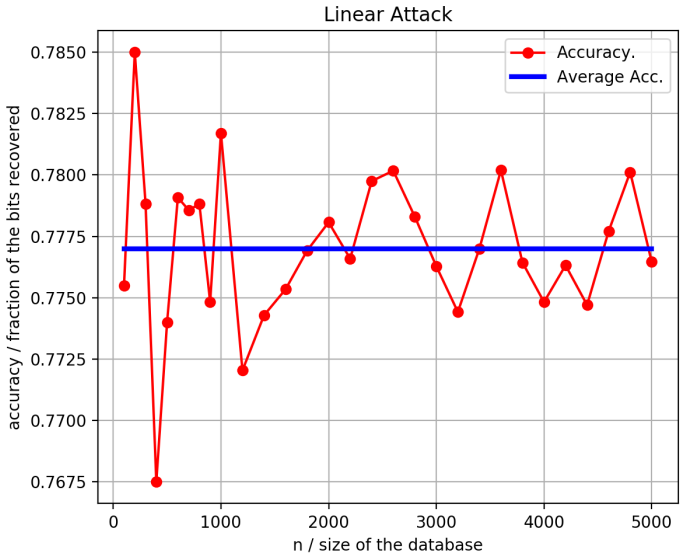


Figure 1: P1 - 1 : accuracy of recontruction attack without aux information

Algorithm 3 Minimizing Error Reconstruction Attack without Extra Information

Require: The observed results a from query, the number of iteration T, the query result release mechanism M.
Initialize dataset s, error e, $s[i] \leftarrow \{0, 1\}$, $e = Inf$.
for $i \in [T]$ **do**.
sample a new data set s'
a' = M(s').
e' = a - a'.
If $e < e'$ **do**. s = s'
return s.

– In the situation where $a[i] - a[i-1] < 0$, instead of just setting $s[i]$ be 0, I went back and retraced the previous result.

These two steps improved the accuracy from the naive method.

- **Code Documentation** See Appendix 1.
- Repository Link
  `https://github.com/jiawenliu/CS591S1.git`

2. The attacker's inputs consist of a and the vector of guesses w.

- **Algorithm Description**
- **Experimental Results Plotting** Figure 2
- **Results Discussion**
- **Code Documentation** The Code are exactly the same as code in previous part, except adding one function in Appendix 2

Algorithm 4 Reconstruction Attack with Extra Information

Require: The observed results a from query, the guass w.
Initialize $c = a$, as the reconstructed counter .
for $i \in [a.length]$ **do**.
let $s = a[i] - c[i-1]$ be the possible j_{th} item in data set.
If $s > 1$ **do**. $c[i] = a[i] - 1$ {the correct counter at position i must be a[i] - 1}
Elif $s < 0$ **do**.
let $j = i$
While $j - 1 > 0 \wedge c[j] < c[j-1]$ **do**. $j-- = 1$ $c[j]-- = 1$
{the correct counter at position i must be a[i] and decrease the previous by 1 until their difference ≥ 0}
return $s_i = c_i - c_{i-1}$ **for** $i \in [a.length]$.

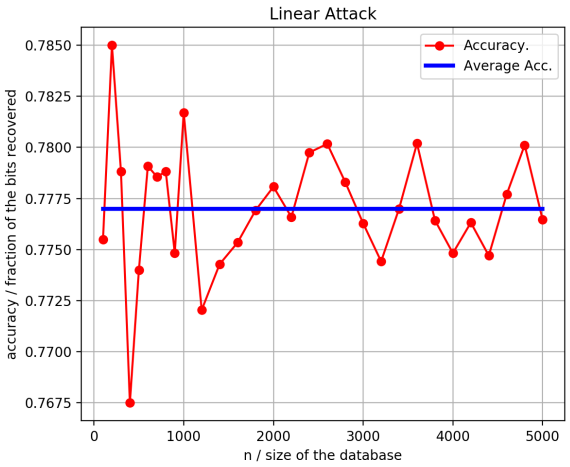


Figure 2: P1 - 2 : accuracy of recontruction attack with aux information

2 Problem 2

1. **(a)** Given $X_i(j)$ sampled from $\{-1, 1\}$ i.i.d for $i = 1, \cdots, n$ and $j = 1, \cdots, d$, we have:

$$E(X_i(j)) = 0,\ Var(X_i(j)) = 1$$
$$E(\bar{X}(j)) = E(\tfrac{1}{n}\textstyle\sum_i X_i(j)) = 0,\ Var(\bar{X}(j)) = Var(\tfrac{1}{n}\textstyle\sum_i X_i(j)) = \tfrac{1}{n}$$

Since $A = M(X) = \bar{X} + (Z(1), \cdots, Z(d))$ where $Z(j) \sim N(0, \sigma^2)$ independently from X, we have:

$$E(A(j)) = 0,\ Var(A(j)) = Var(\bar{X}(j) + Z(j)) = \frac{1}{n} + \sigma^2$$

Since T_{out} is independently from A, we can conclude the expectation of $F(A, T_{out})$:

$$E(F(A, T_{out})) = E(\sum_{j=1}^{d} A(j) * T_{out}(j)) = \sum_{j=1}^{d} E(A(j) * T_{out}(j)) = 0$$

and the standard deviation:

$$\begin{aligned}
& \sqrt{Var(F(A, T_{out}))} \\
= & \sqrt{Var(\sum_{j=1}^{d} A(j) * T_{out}(j))} \\
= & \sqrt{\sum_{j=1}^{d} Var(A(j) * T_{out}(j))} \\
= & \sqrt{\sum_{j=1}^{d} Var(A(j)) * Var(T_{out}(j)) + Var(A(j)) * E(T_{out}(j))^2 + E(A(j))^2 * Var(T_{out}(j))} \\
= & \sqrt{d(\tfrac{1}{n} + \sigma^2)}
\end{aligned}$$

(b) Since T_{in} is sampled from X which is not independently from A anymore, we have following:

$$\begin{aligned}
F(A, T_{in}) &= \sum_{j=1}^{d} A(j) * T_{in}(j) \\
&= \sum_{j=1}^{d} (\bar{X}(j) + Z(j)) * T_{in}(j) \\
&= \sum_{j=1}^{d} (\tfrac{1}{n} \sum_{i=1}^{n} X_i(j) + Z(j)) * T_{in}(j) \\
&= \sum_{j=1}^{d} (\tfrac{1}{n} (\sum_{i=1 \ldots n \wedge i \neq in} X_i(j) + Z(j)) * T_{in}(j) + \tfrac{1}{n} T_{in}(j)^2)
\end{aligned}$$

Let $A_{/T_{in}} = \frac{1}{n}(\sum_{i=1 \ldots n \wedge i \neq in} X_i(j) + Z(j))$, we have:

$$E(A_{/T_{in}}) = 0, \ Var(A_{/T_{in}}) = (\frac{n-1}{n^2} + \sigma^2)$$

So we have the expectation of $F(A, T_{in})$ as:

$$E(F(A, T_{in})) = \sum_{j=1}^{d} E((A_{/T_{in}} * T_{in}(j) + \frac{1}{n} T_{in}(j)^2)) = \frac{d}{n}$$

and variance of $F(A, T_{in})$ as:

$$\begin{aligned}
Var(F(A, T_{in})) &= \sum_{j=1}^{d} Var((A_{/T_{in}} * T_{in}(j) + \tfrac{1}{n} T_{in}(j)^2)) \\
&= \sum_{j=1}^{d} Var((A_{/T_{in}} * T_{in}(j)) \\
&= d(\tfrac{n-1}{n^2} + \sigma^2)
\end{aligned}$$

The standard deviation is the square root of variance, i.e., $\sqrt{d(\frac{n-1}{n^2} + \sigma^2)}$.

(c)

$$d = \left(\sqrt{(n + n^2\sigma^2)} + \sqrt{(n - 1 + n^2\sigma^2)} \right)^2$$

- $\sigma = 0.1$

$$d = \left(\sqrt{(n + 0.01n^2)} + \sqrt{(n - 1 + 0.01n^2)} \right)^2$$

- $\sigma = 1/n$

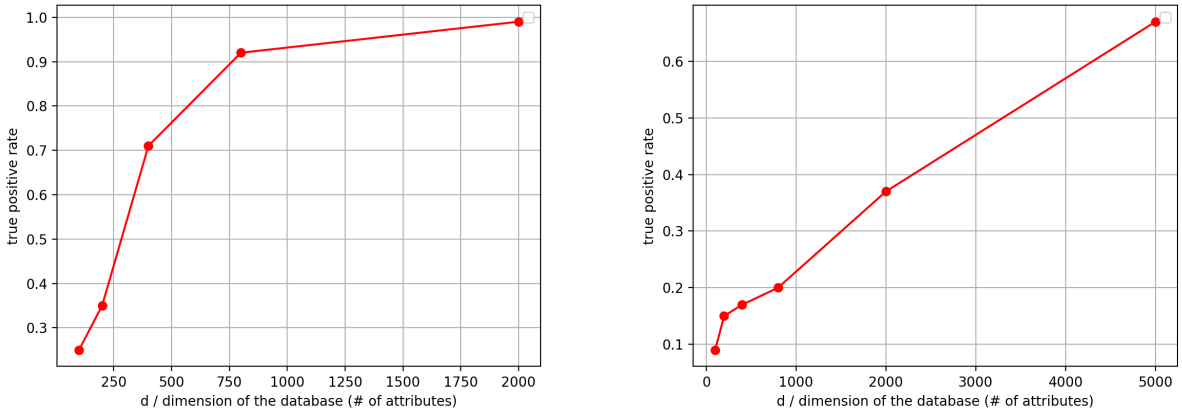
$$d = \left(\sqrt{(n+1)} + \sqrt{(n)} \right)^2$$

- $\sigma = 1/\sqrt{n}$

$$d = \left(\sqrt{2n} + \sqrt{2n-1} \right)^2$$

Algorithm 5 Inference Attack with Gaussi Mechanism

Require: The row and column # of data base d, n, the parameter σ for Guassi mechanism.
Sample $d_{in}, d_{out} \leftarrow \{-1, 1\}^{d \times n}$.
let query $q(d) = map(sum, d.transpose)/n$
let $A = q(d_{in}) + [z_i \leftarrow N(0, \sigma) for\ i \in [d]]$
let $score_{in}, score_{out} = Ad_{in}.transpose, Ad_{out}.transpose$
{the F score for each row in dataset in and dataset out.}
let $c =$ Counting how many items in $score_{in}$ greater than all items in $score_{out}$.
let $p = c/n$
return True positive p



(a) $\sigma = 0.01$ (b) $\sigma = 1/3$

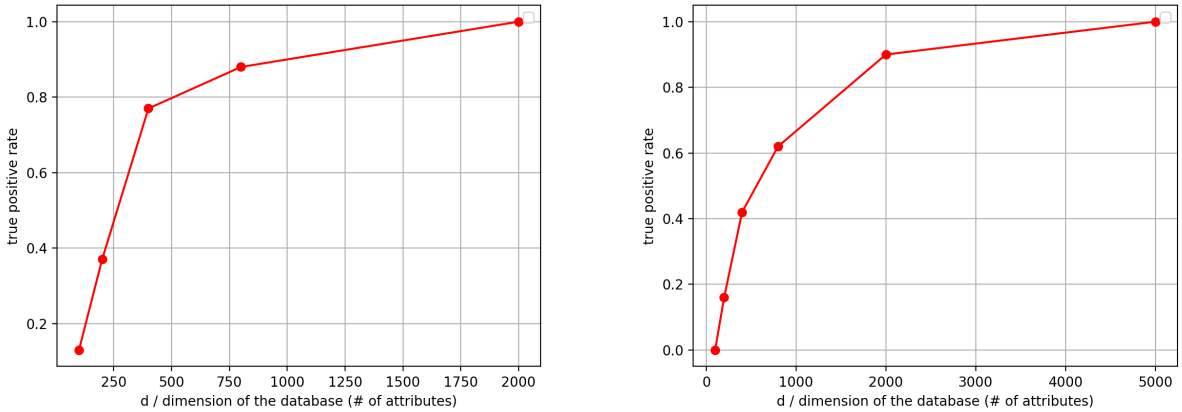
Figure 3: P2 : Gaussian mechanism

2. **(a)**
   - **Algorithms Description** The Alg. 5
   - **Experimental Results Plotting** Figure 3
   - **Results Discussion** Repeating the Algorithm. 5, we get the results poltted in Fig. 3.
     The Fig. 3(a) shows the True positive rate with guassi noise parameter $\sigma = 0.01$. We can see the accuracy achieved 1.0 when data base columns number is 2000. Actually already achieved 0.92 when column number greater than 800.
     The Fig. 3(b) shows the True positive rate with guassi noise parameter $\sigma = 1/3$. We can see the accuracy only achieved 0.38 when data base columns number is 2000. Even though the column number increase up to 5000, accuracy only get to around 0.68.
     From the two comparison, we can tell the smaller the parameter σ is, the more accuracy it is. This is consistent with our intuition that the smaller noise results in more accuracy.
   - **Code** See Appendix 3

   (b)
   - **Algorithms Description** The Alg. 6
   - **Experimental Results Plotting** Figure 4
   - **Results Discussion**

Algorithm 6 Inference Attack with Rounding Mechanism

Require: The row and column # of data base d, n, the parameter σ for Guassi mechanism.
Sample $d_{in}, d_{out} \leftarrow \{-1, 1\}^{d \times n}$.
let query $q(d) = map(sum, d.transpose)/n$
let $A = round(q(d_{in}))$
let $score_{in}, score_{out} = A\dot{d}_{in}.transpose, A\dot{d}_{out}.transpose$
{the F score for each row in dataset in and dataset out.}
let $c =$ Counting how many items in $score_{in}$ greater than all items in $score_{out}$.
let $p = c/n$
return True positive p



(a) $\sigma = 0.01$ (b) $\sigma = 1/3$

Figure 4: P2 : rounding mechanism

Repeating the Algorithm. 5 with different column number, we get the results poltted in Fig. 4.

The Fig. 4(a) shows the True positive rate with guassi noise parameter $\sigma = 0.01$. We can see the accuracy achieved 1.0 when data base columns number is 2000. Actually already achieved 0.92 when column number greater than 800.

The Fig. 4(b) shows the True positive rate with guassi noise parameter $\sigma = 1/3$. We can see the accuracy only achieved 0.9 when data base columns number is 2000. However, when the column number increase up to 5000, accuracy only get to around 1.0.

From the two plots for rounding mechanism, we can tell the smaller the parameter σ is, the more accuracy it is. This is consistent with our intuition that the smaller noise results in more accuracy.

By comparison with the Guassi Mechanism, we can obtain that the in small (i.e. 0.01) noise parameter, the guassi mechanism can achieve better accuracy. But in larger noise mechanism (i.e. 1/3) rounding mechanism is able to get a better accuracy.

- **Code** See Appendix 4.

3. - **Algorithms Description** Utilizing the library function and simply compare if the products are greater than a threshold, the algorithm shown in Alg. 7.

Algorithm 7 Linear Regression Attack

Require: The row and column # of data base d, n, the parameter σ for Guassi mechanism.
Sample $d, d_{out} \leftarrow \{-1, 1\}^{d \times n}$.
Sample $coff \leftarrow \{-1/math.sqrt(d), 1/math.sqrt(d)\}^d$.
let label $y, y_{out} = genlabel(d, coff), genlabel(d, coff)$.
let $classifier(w, target) = 1.0\ if\ abs(w \cdot target[0]^T - target[1]) < 0.01\ else\ 0.0$
let $score_{in}, score_{out} = classifier(w, d), classifier(w, d_{out})$
let $tc =$ Counting # items in $score_{in}$ is 1, $fc =$ Counting # items in $score_{out}$ is 0.
let $tp = tc/n$, $fp = fc/n$
return True positive tp and True negative fp

- **Experimental Results Plotting** Figure 5.

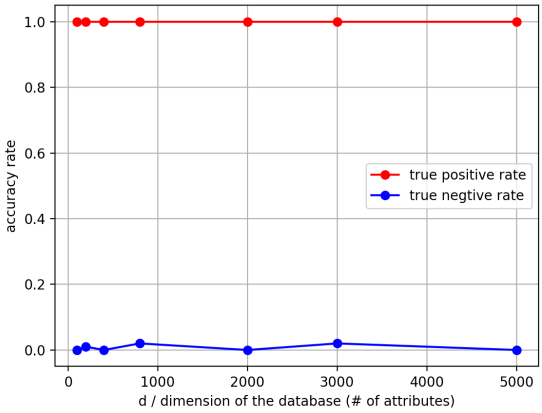


Figure 5: P3: True Positive and True Negtive of Linear Regression Attack

- **Results Discussion** Close to perfect.
- **Code** See Appendix 5

3 Problem 3

1. (reading)
2. (Sampling)
   (a) Given S_i sampled from $a_1, \ldots, a_n$, we have:

$$E(S_i) = \bar{a}.$$

Let $E(S) = E(\sum S_i) = k\bar{a}$, we have:

$$\begin{aligned} Pr[error \leq \alpha] &= Pr[|\hat{S} - \bar{a}| \leq \alpha] \\ &= Pr[|\tfrac{1}{k}\sum S_i - \bar{a}| \leq \alpha] = Pr[|\sum S_i - k\bar{a}| \leq k\alpha] \\ &= Pr[|S - E(S)| \leq k\alpha] \\ &\geq 1 - 2\exp\big(-\tfrac{\alpha^2/\bar{a}^2}{2+\alpha/\bar{a}}k\bar{a}\big) \text{(applying the Chernoff bounds)} \end{aligned}$$

By guaranteeing $\delta = 2\exp\big(-\frac{\alpha^2/\bar{a}^2}{2+\alpha/\bar{a}}k\bar{a}\big)$, we have:

$$k = \frac{\ln(\frac{2}{\delta})(2+\frac{\alpha}{\bar{a}})\bar{a}}{\alpha^2}$$

(b) let μ_i be the true proportion of individuals in the population who answer "yes" for function f_i, i.e. $\frac{1}{n}\sum_{j=1}^{n} f_i(x_j) = \mu_i$. We know $E[f_i(S_j)] = \mu_i$ and $E[f_i(S)] = k\mu_i$. Let $f_i(S) = \sum_{j=1}^{k} f_1(S_j)$, we have for all f_i:

$$\begin{aligned}
& Pr[|\tfrac{1}{k}f_1(S) - \mu_1| \le \alpha \wedge \ldots \wedge |\tfrac{1}{k}f_d(S) - \mu_d| \le \alpha] \\
= \;& 1 - Pr[|\tfrac{1}{k}f_1(S) - \mu_1| \ge \alpha \vee \ldots \vee |\tfrac{1}{k}f_d(S) - \mu_d| \ge \alpha] \\
= \;& 1 - Pr[|f_1(S) - k\mu_1| \ge k\alpha \vee \ldots \vee |f_d(S) - k\mu_d| \ge k\alpha] \\
\ge \;& 1 - \sum_{i=1}^{d} Pr[|f_1(S) - k\mu_1| \ge k\alpha] \quad \text{(applying the union bound)} \\
\ge \;& 1 - 2\sum_{i=1}^{d} \exp(-\tfrac{\alpha^2/\mu_i}{2+\alpha/\mu_i}k\mu_i) \quad \text{(applying the Chernoff bound)} \\
= \;& 1 - 2\sum_{i=1}^{d} \exp(-\tfrac{k\alpha^2}{2\mu_i+\alpha}) \\
\ge \;& 1 - 2d\exp(-\tfrac{k\alpha^2}{2+\alpha})
\end{aligned}$$

To guaranteeing the $\delta = 2d\exp(-\frac{k\alpha^2}{2+\alpha})$, we have:

$$d = \frac{\ln(2d\frac{1}{\delta})(2+\alpha)}{\alpha^2} = \frac{(\ln(2d)+\ln(\frac{1}{\delta}))(2+\alpha)}{\alpha^2} = O(\frac{\ln(2d)+\ln(\frac{1}{\delta})}{\alpha^2})$$

(c) Given $err = \frac{Hamming(\hat{x},x)}{n}$, i.e.

$$err = \frac{1}{n}||\hat{x} - x||_{l_1} = \frac{1}{n}\sum_{i=1}^{n}|\hat{x}(i) - x(i)|$$

Let x' be the sub data set which is not used to answer mechanism, and $\hat{x'}$ be the gauss of this subset. The best the adversary can do is perfectly recover the subset used $(x \setminus x')$ and randomly guess x', so we have:

$$E(err) = \frac{1}{n}(\frac{2n}{3}E(|\hat{x'}(i) - x'(i)|) + \frac{n}{3} * 0) = \frac{1}{3}$$

By applying the Chernoff bound, we have:

$$Pr[err \ge (1+\epsilon)\frac{1}{3}] \le 1 - \exp(-\frac{\epsilon^2}{2+\epsilon}\frac{n}{3})$$

Given $\Omega(n) \sim \exp(-\frac{\epsilon^2}{2+\epsilon}\frac{n}{3})$, we have:

$$err \ge (1+\epsilon)\frac{1}{3} \ge \frac{1}{3} > \frac{1}{4}$$

4 Problem 4

1. (Sample Size Calculations.)

   (a) Given $X_i \sim Bernoulli(p)$, we have:

   $$E(X_i) = p, \quad Var(X_i) = p(1-p).$$

   Since $\bar{X}$ is the mean of i.i.d. $X_1, \cdots, X_n$, we have:

   $$E(\bar{X}) = p, \quad Var(\bar{X}) = \frac{1}{n}p(1-p).$$

   To guarantee that variance of $\bar{X} < \alpha$, the following bound on n can be derived by:

   $$\begin{array}{rcl} Var(\bar{X}) = \frac{1}{n}p(1-p) & < & \alpha \\ n & > & \frac{p(1-p)}{\alpha} \ (\star). \end{array}$$

   Then we have $n_\epsilon(\alpha, p)$ be the ceil of $(\star)$:

   $$n_\epsilon(\alpha, p) = \left\lceil \frac{p(1-p)}{\alpha} \right\rceil.$$

   Given $A_\epsilon(X) = \bar{X} + Z$ where $Z \sim Lap(1/n\epsilon)$, we have:

   $$Var(A_\epsilon(X)) = Var(\bar{X}) + Var(Z) = \frac{1}{n}p(1-p) + \frac{2}{(n\epsilon)^2}$$

   Similarly, to guarantee that variance of $\bar{X} < \alpha$, the following bound on n^* can be derived by:

   $$\begin{array}{rcl} Var(A_\epsilon(X)) = \frac{1}{n}p(1-p) + \frac{2}{(n\epsilon)^2} & < & \alpha \\ n & > & \frac{p(1-p)}{\alpha - \frac{2}{(n\epsilon)^2}} \ (\diamond). \end{array}$$

   Then we have $n_\epsilon(\alpha, p)$ be the ceil of $(\star)$:

   $$n^*_\epsilon(\alpha, p) = \left\lceil \frac{p(1-p)}{\alpha - \frac{2}{(n\epsilon)^2}} \right\rceil$$

   (b)

   $$n_\epsilon(\alpha, p) - n^*_\epsilon(\alpha, p) = p(1-p)\big(\frac{1}{\alpha} - \frac{1}{\alpha - 2/(n\epsilon)^2}\big)$$

   Since

   $$\big(\frac{1}{\alpha} - \frac{1}{\alpha - 2/(n\epsilon)^2}\big) \left\{ \begin{array}{ll} > 0 & \alpha < \frac{1}{\alpha - 2/(n\epsilon)^2} \\ < 0 & \alpha > \frac{1}{\alpha - 2/(n\epsilon)^2} \end{array} \right\},$$

   we have the value of $n_\epsilon(\alpha, p) - n^*_\epsilon(\alpha, p)$:

   $$\left\{ \begin{array}{ll} \left\{ \begin{array}{ll} \text{decrease} & p \in [0.5, 1] \\ \text{increase} & p \in [0, 0.5) \end{array} \right\} & \alpha < \frac{1}{\alpha - 2/(n\epsilon)^2} \\ \left\{ \begin{array}{ll} \text{increase} & p \in [0.5, 1] \\ \text{decrease} & p \in [0, 0.5) \end{array} \right\} & \alpha > \frac{1}{\alpha - 2/(n\epsilon)^2} \end{array} \right\},$$

(c) When A_ϵ is the ϵ-DP randomized response estimator, we have:

$$q_\epsilon(X_i) = \left\{ \begin{array}{ll} X_i & w.p.\ \frac{e^\epsilon}{1+e^\epsilon} \\ 1-X_i & w.p.\ \frac{1}{1+e^\epsilon} \end{array} \right\},\ A_\epsilon(X) = \frac{1}{n}\sum_i q_\epsilon(X_i)$$

we have the expectation and variance for $q_\epsilon(X_i)$ as:

$$\begin{array}{ll} E(q_\epsilon(X_i)) & = E(\frac{e^\epsilon}{1+e^\epsilon}X_i + \frac{1}{1+e^\epsilon}(1-X_i)) = \frac{1}{1+e^\epsilon}((e^\epsilon-1)E(X_i)+1) = \frac{1}{1+e^\epsilon}((e^\epsilon-1)p+1) \\ Var(q_\epsilon(X_i)) & = E\big((q_\epsilon(X_i) - E(q_\epsilon(X_i)))^2\big) = E\big(\frac{e^\epsilon}{1+e^\epsilon}\big(X_i - E(q_\epsilon(X_i))\big)^2 + \frac{1}{1+e^\epsilon}\big((1-X_i) - E(q_\epsilon(X_i))\big)^2 \end{array}$$

Let $E_{q_\epsilon} = E(q_\epsilon(X_i))$, we have:

$$\begin{array}{ll} Var(q_\epsilon(X_i)) & = \frac{1}{1+e^\epsilon}E((1+e^\epsilon)(X_i^2 + E_{q_\epsilon}^2) + 1 - 2E_{q_\epsilon} + 2(1-e^\epsilon)X_i E_{q_\epsilon}) \\ & = (p + E_{q_\epsilon}^2) + \frac{1}{1+e^\epsilon}(1 - 2E_{q_\epsilon} + 2(1-e^\epsilon)pE_{q_\epsilon}) \\ & = (\frac{1}{1+e^\epsilon} + p - E_{q_\epsilon}^2) \end{array}$$

Then we have:

$$\begin{array}{ll} E(A_\epsilon(X)) & = E\big(\frac{1}{n}\sum_i q_\epsilon(X_i)\big) = \frac{1}{1+e^\epsilon}((e^\epsilon-1)p+1) \\ Var(A_\epsilon(X)) & = \frac{1}{n}Var\big(q_\epsilon(X_i)\big)\big) = \frac{1}{n}(\frac{1}{1+e^\epsilon} + p - E_{q_\epsilon}^2) \end{array}$$

To guarantee $Var(A_\epsilon(X)) > \alpha$, we have:

$$n^*_{\alpha,p} > \frac{\frac{1}{1+e^\epsilon} + p - E_{q_\epsilon}^2}{\alpha}$$

2. (Global Sensitivities.)

(a) 1.0

(b) 1.0

(c) $\frac{1}{n}$

(d) 1.0

(e) $(n-1)$, where n is the size of the Census data (n is not explicitely noted).

(f) 1.0

(g) ∞

(h) 1.0

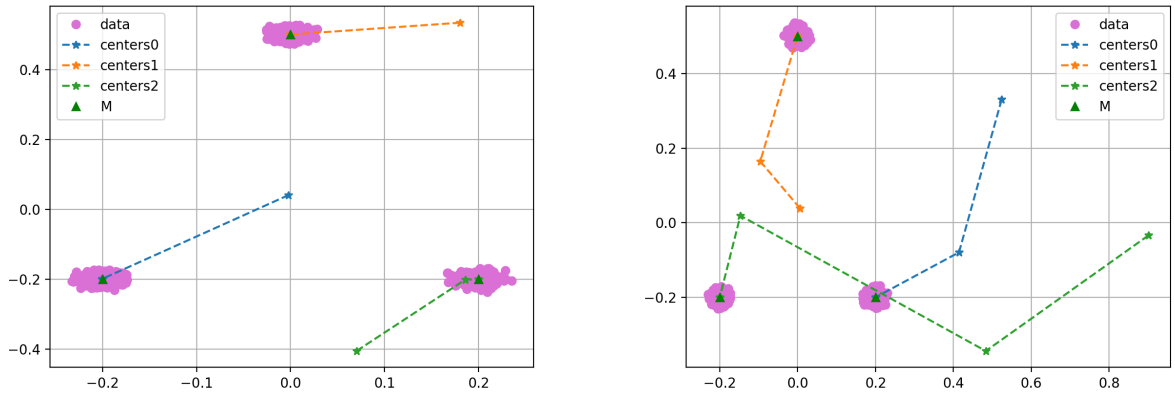
3. (Amplification by subsampling) $B_{p,\epsilon}$ is $\ln(1+(e^\epsilon-1)p)$-differentially private.

Proof. Given A_ϵ is ϵ-DP, we have for arbitrary adjacent data set x, x' and G be the subset of output domain:

$$e^{-\epsilon} \le \frac{Pr[A_\epsilon(x) \in G]}{Pr[A_\epsilon(x') \in G]} \le e^\epsilon$$

So we have for $B_{p,\epsilon}$ under the same setting where x, x' differ in x_k and S is the subset:

$$\begin{array}{lll} \frac{Pr[B_{p,\epsilon}(x')\in G]}{Pr[B_{p,\epsilon}(x)\in G]} & = & \frac{Pr[A_\epsilon(S)\in G \wedge x_k \notin S \cup A_\epsilon(S')\in G \wedge x_k \in S]}{Pr[A_\epsilon(S)\in G]} \\ & \le & \frac{Pr[A_\epsilon(S)\in G \wedge x_k \notin S] + Pr[A_\epsilon(S')\in G \wedge x_k \in S]}{Pr[A_\epsilon(S)\in G]} \\ & = & \frac{(1-p)Pr[A_\epsilon(S)\in G] + pPr[A_\epsilon(S')\in G]}{Pr[A_\epsilon(S)\in G]} \\ & \le & \frac{(1-p)Pr[A_\epsilon(S)\in G] + pe^\epsilon Pr[A_\epsilon(S)\in G]}{Pr[A_\epsilon(S)\in G]} \\ & = & 1 + (e^\epsilon - 1)p \end{array}$$



(a) Pure K-Means

(b) Private K-Means

Figure 6: P4: K-Means Algorithms

The other side can be proved trivially. Then we have $B_{p,\epsilon}$ is $\ln(1+(e^{\epsilon}-1)p)$-DP.

□

4. (K-means algorithm)

- **Algorithm Description** The algorithm is the same as algorithm presented in class

Algorithm 8 Differentially Private K-means

Require: The rows and columns of the data points n, d, the parameters k, T, ϵ, Centers M and parameter σ for generating data points.
Generate data points $x_i = M + (z \leftarrow N(0,\sigma)^d)$
let $k = M.length$, $\epsilon' = \frac{\epsilon}{2T}$.
Initialize c randomly from $[-1,1]^{d\times k}$.
For $t \in [T]$ **do**.
$s_j = \{i : c_j$ is the closet center to $x_i\}$.
$n_j = |s_j| + Lap(\frac{1}{\epsilon'})$
$a_j = \sum_{i\in s_j} x_i + Lap(\frac{2}{\epsilon'})^d$
$c_j^t = \frac{a_j}{n_j}$ if $n_j > 5$ else randomly from $[-1,1]^{d\times k}$.
return c.

- **Experimental Results Plotting** Figrue 6.
- **Results Discussion** The results are shown in Fig. 6.
  In Fig. 6(a), I firstly implemented the pure k-means algorithm which performs very well with only 2 or even 1 iteration arrived the true center.
  Then **in Fig. 6(b)**, the differentially private k-means algorithm performs also good but with few more iteration in order to arrive the true center.
  During my experiments, some times the algorithm doesn't converge to a final center because of the randomness.
- **Code Documentation** See Appendix 6

Appendix

```
import random
import numpy as np

#GENERATING DATA SIZE AND CONRRESPONDING PARAMETER
def gen_dataset(n):
  return [random.randint(0, 1)for i in range(n)]
def gen_datasizes(r, step):
  return [i*step for i in range(r[0]/step,r[1]/step + 1)]

#RELEASING THE NOIZED VERSION OF DATABASE
def releasing_dataset(dataset):
  return [sum(dataset[:(i + 1)]) + random.randint(0,1) for i in range(len(dataset)
    )]

#ATTACK WITH ONLY THE KNOWLEDGE OF THE OBSERVATION OF ONE DATABASE
def attack_no_aux(observation):
  rec_counter = [observation[0] - 1 if observation[0] > 1 else observation[0]]
  for i in range(1, len(observation)):
    s = observation[i] - rec_counter[i - 1]
    if s > 1:
      rec_counter.append(observation[i] - 1)
    elif s < 0:
      rec_counter.append(observation[i])
      j = i
      while j - 1 >= 0 and rec_counter[j] < rec_counter[j - 1]:
        j -= 1
        rec_counter[j] -= 1
    else:
      rec_counter.append(observation[i])
  return np.array([rec_counter[0]] + [rec_counter[i] - rec_counter[i - 1] for i in
     range(1, len(observation))])

def accuracy(att, data):
  return sum([1 if att[i] == data[i] else 0 for i in range(len(att))])/ (len(att)
    *1.0)
def plot_accuracy(ys, ns):
  plt.figure()
  plt.plot(ns, ys, "ro-", label = "Accuracy.")
  plt.plot(ns,[sum(ys)/len(ys)]*len(ys), "b-", label="Average Acc.",linewidth=3.0)
  plt.xlabel("n / size of the database")
  plt.ylabel("accuracy / fraction of the bits recovered")
  plt.title("Linear Attack")
  plt.legend()
  plt.grid()
  plt.show()

#EXPERIMENTING WITH FIXED N FOR K ROUNDS
def exprmt_k(n, k):
  acc = 0.0
  for i in range(k):
    dataset, q = gen_dataset(n), gen_query(n)
    acc += accuracy(attack_no_aux(releasing_dataset(q,dataset)),dataset)
  return acc/k
def exprmt_k_ns(ns, k):
  return [testing_kround(n, k) for n in ns]

```

```
if __name__ == "__main__":
  datasizes = gen_datasizes((100,900),100)+gen_datasizes((1000,5000),200)+[50000]
  plot_accuracy(exprmt_k_ns(datasizes, 20), datasizes)
```

Listing 1: Python Code For Problem 1 - 1

```
def attack_with_aux(observation, guess):
  attack_no_aux(observation), w
  return
```

Listing 2: Python Code For Problem 1 - 2

```
\label{p2-2a}
\input{p2-2a.tex}
import numpy as np
import matplotlib.pyplot as plt
import random
#GENERATING DATA SIZE AND CONRRESPONDING PARAMETER
def gen_data(d, n): return np.array([[random.choice([-1,1]) for _ in range(d)] for
     i in range(n)])

def F_test(A, X): return np.dot(A, X.transpose())

def query_avg(data):
  return np.array(map(sum, data.transpose()))/(len(data)*1.0)

def gaussi_mech(query, sigma, data): return query(data) + np.array([random.gauss
    (0, sigma) for _ in data[0]])
def scores(A, data): return F_test(A, data)

def true_positive(sc1, sc2):
  return sum(map(lambda s: 1.0 if s*1.0/len(sc1) > 0.95 else 0,
    [sum(map(lambda s2: 1.0 if s1 > s2 else 0, sc2)) for s1 in sc1])) / len(sc1)

def exprmt(d, n, sigma):
  p, k = 0.0, 1
  for i in range(k):
    data, out_data = gen_data(d, n), gen_data(d, n)
    A = gaussi_mech(query_avg, sigma, data)
    p += true_positive(scores(A, data), scores(A, out_data))
  return p/k

def exprmt_d(n, d_list, sigma): return [exprmt(d, n, sigma) for d in d_list]

def plot_accuracy(ys, ns):
  plt.figure()
  plt.plot(ns, ys, "ro-")
  plt.xlabel("d / dimension of the database (# of attributes)")
  plt.ylabel("true positive rate")
  plt.legend()
  plt.grid()
  plt.show()

if __name__ == "__main__":
  d_list = [100, 200, 400, 800, 2000, 5000]
  tp = exprmt_d(100, d_list, 1/3.0)
  plot_accuracy(tp, d_list)
```

Listing 3: Python Code for Problem 2 - 2 - (a)

```python
def rounding_mech(query, sigma, data):
  return np.array(map(round, query(data)/sigma)) * sigma
```

Listing 4: Python Code for Problem 2 - 2 - (b)

```python
import numpy as np
import matplotlib.pyplot as plt
import random
import math
from functools import partial

#GENERATING DATA SIZE AND CONRRESPONDING PARAMETER
def gen_data(d, n):
  return np.array([[random.choice([-1,1]) for _ in range(d)] for i in range(n)])

def gen_cfft(d):
  return np.array([random.choice([-1/math.sqrt(d), 1/math.sqrt(d)]) for _ in range
    (d)])

def gen_label(data, c, sigma):
  return np.dot(c, data.transpose()) + np.array([random.gauss(0, sigma) for _ in
    data])

def LSLR(data, label):
  return np.linalg.lstsq(data, label)[0]

def classifier(w, target):
  return 1.0 if abs(np.dot(w, target[0].transpose()) - target[1]) < 0.01 else 0.0

def scores(w, pos_targets, neg_targets):
  return [classifier(w, p) for p in pos_targets], [classifier(w, p) for p in
    neg_targets]

def true_positive(sc1, sc2):
  return sum(sc1) / (len(sc1))

def false_negtive(sc1, sc2):
  return 1.0 - sum(sc1) / (len(sc1))

def true_negtive(sc1, sc2):
  return sum(sc2) / (len(sc2))

def false_positive(sc1, sc2):
  return 1.0 - sum(sc2) / (len(sc2))

def exprmt(d, n, sigma):
  data, coefft, neg_data = gen_data(d, n), gen_cfft(d), gen_data(d, n)
  y, neg_y = gen_label(data, coefft, sigma), gen_label(neg_data, coefft, sigma)
  score = scores(LSLR(data, y), zip(data, y), zip(neg_data, neg_y))
  return true_positive(score[0], score[1]), true_negtive(score[0], score[1])

def exprmt_d(n, d_list, sigma):
  return [exprmt(d, n, sigma) for d in d_list]

def plot_accuracy(ys, ns):
  plt.figure()
  plt.plot(ns, [y[0] for y in ys], "ro-", label = "true positive rate")
  plt.plot(ns, [y[1] for y in ys], "bo-", label = "true negtive rate")
  plt.xlabel("d / dimension of the database (# of attributes)")
```

```
  plt.ylabel("accuracy rate")
  plt.legend()
  plt.grid()
  plt.show()

if __name__ == "__main__":
  d_list = [100, 200, 400, 800, 2000, 3000, 5000]
  tp = exprmt_d(100, d_list, 0.1)
  plot_accuracy(tp, d_list)
```

Listing 5: Python Code for Problem 2 - 3

```
import numpy as np
import matplotlib.pyplot as plt
import random

#GENERATING DATA SIZE AND CONRRESPONDING PARAMETER
def gen_data(d, n,  M, sigma):
  unnormx = [random.choice(M) + np.array([random.gauss(0, sigma) for _ in range(d)
    ]) for i in range(n)]
  return np.array([ x / sum(abs(x)) if sum(abs(x)) > 1 else x for x in unnormx])

def gen_point(d):
  p = np.array([random.uniform(-1, 1) for _ in range(d)])
  while sum(abs(p)) > 1.0:
    p = np.array([random.uniform(-1, 1) for _ in range(d)])
  return np.array(p)

#PURE K_MEAN ALGORITHM
def k_clustering(data, T, eps, k):
  ctrs = np.array([gen_point(len(data[0])) for _ in range(k)]) #initialize the
    centers
  ctrs_record = []
  for _ in range(T):
    distance, S, used = [[sum(abs(row - c)) for c in ctrs] for row in data], [[]
    for _ in range(k)], [False]*len(data)
    for j in range(k):
      for i in range(len(data)):
        if distance[i][j] <= min(distance[i]) and not used[i]:
          used[i] = True
          S[j].append(data[i])
    ctrs = [sum(S[i])/ len(S[i]) if S[i] != [] else gen_point(len(data[0])) for i
    in range(k)]
    ctrs_record.append(np.array(ctrs))
  return np.array(ctrs), np.array(ctrs_record)

#PRIVATE VERSION OF K_MEAN ALGORITHM
def k_clustering_private(data, T, eps, k):
  ctrs = np.array([gen_point(len(data[0])) for _ in range(k)]) #initialize the
    centers
  eps, ctrs_record = eps/(2.0 * T), []
  for _ in range(T):
    distance, S, used = [[sum(abs(row - c)) for c in ctrs] for row in data], [[]
    for _ in range(k)], [False]*len(data)
    for j in range(k):
      for i in range(len(data)):
        if distance[i][j] <= min(distance[i]) and not used[i]:
          S[j].append(data[i])
          used[i] = True # prevent two data point added to two different sets when
```

```
     they have the same distance
    ctrs = [(sum(S[i]) + np.random.laplace(0.0, eps))/ (len(S[i]) + np.random.laplace(0.0, 2*eps)) if len(S[i]) > 5 else gen_point(len(data[0])) for i in range(k)]
     ctrs_record.append(np.array(ctrs))
  return np.array(ctrs), np.array(ctrs_record)


def plot_accuracy(data, centers, M):
  def obx (data):
    return list(data.transpose()[0])
  def oby (data):
    return list(data.transpose()[1])
  def ithcx(data, i):
    return data.transpose()[0][i]
  def ithcy(data, i):
    return data.transpose()[1][i]
  plt.figure()
  plt.plot(obx(data), oby(data), "o", color="orchid", label="data")
  for i in range(len(M)):
    plt.plot((ithcx(centers, i)), (ithcy(centers, i)), "*--", label="centers" + str(i))
  plt.plot(obx(M), oby(M), "g^", label="M")
  plt.legend()
  plt.grid()
  plt.show()




if __name__ == "__main__":

  #SETTING UP THE PARAMETERS WHEN DOING GROUPS EXPERIMENTS
  M = np.array([[0, 0.5], [0.2, -0.2], [-0.2, -0.2]])
  data = gen_data(2, 2000, M, 0.01)
  #EXPERIMENTS WIEH PRIVATE K_MEANS
  centers, records = k_clustering_private(data, 10, 0.1, 3)
  plot_accuracy(data, records, M)
  #EXPERIMENTS WIEH PURE K_MEANS
  centers, records = k_clustering(data, 10, 0.1, 3)
  plot_accuracy(data, records, M)
```

Listing 6: Python Code for Problem 4 - 3