南開大學

《数据安全》课程实验报告

实验五：交互式发布 DP 方案评估

学　　院　网络空间安全学院

专　　业　信息安全

学　　号　2112060

姓　　名　孙蕗

目录

一、实验要求

参考教材实验5.1.对交互式发布方案进行DP方案设计，指定隐私预算为0.1，支持查询次数为20次，对DP发布后的记过进行评估说明隐私保护的效果。

二、实验原理

1. 差分隐私

差分攻击的核心即是通过寻找两个仅相差一条记录的数据集，对其分别做同样的查询，再比较返回结果的差异，从而获取两个集合所相差的记录的敏感信息。

为了抵御差分攻击，最直观的想法是通过对查询结果加入一定的扰动（即反馈的查询结果并非是真实的结果），使得查询结果不再精确，攻击者在进行差分攻击时获得的查询结果无法用于区分只差一条记录的两个集合。但如果不对疏浚做任何处理，攻击者可以通过有限次的查询获取个体的隐私信息形成差分攻击。

为了抵御差分攻击，可以在查询结果上增加一个随机数，使得攻击者每次得到的结果都是含有一个随机值的，以此来扰乱攻击者得到的查询结果。这个随机数即可称之为噪音(noise)，其一般是遵循某种分布产生的随机数。在添加噪音后，可以得到新的 $f'(i) = count(i) + noise$。添加的噪音是遵循某种分布均匀随机的，故两次查询得到的结果也是在两个值域上均匀随机的，攻击者将有概率无法再通过两次查询的差值来获取隐私信息，达到一定程度上保护隐私的目的，即能够提供差分隐私。

定义 $x \in X$ 为域 X 中的元素，从 X 中抽取 n 个元素集合组成数据集 D，其中属性的个数为维度 d。X^n表示数据集 D 的集合，即从 X 中抽取 n 个元素组成的数据集的集合。若两个数据集 D 和 D′具有相同的属性结构，二者之间只有一条记录不同，则称 D 和 D′为相邻数据集（Neighboring Datasets）。设算法 M 为一个随机算法，Y 为随机算法的输出域。

差分隐私定义：若随机算法 $M: X^n \to Y$，对于任意子集 $S \subseteq Y$，在任意两个相邻数据集 $D, D' \in X^n$上，满足

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr\left[M\left(D'\right) \in S\right] + \delta$$

则称随机算法 M 满足(ε, δ) −差分隐私。

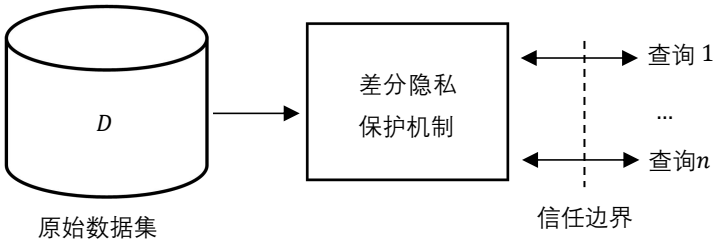
参数δ即表示随机算法 M 不满足差分隐私的概率，通常被设置为可忽略的极小值，$\delta \in [0,1)$。当$\delta = 0$ 时，称随机算法满足$\varepsilon -$差分隐私。$\varepsilon -$差分隐私又称为纯差分隐私（Pure Differential Privacy），而$(\varepsilon, \delta) -$差分隐私又称为近似差分隐私（Approximate Differential Privacy）。

在$\varepsilon -$差分隐私中，我们知道$e^{-\varepsilon} \leq \frac{\Pr[M(D) \in S]}{\Pr[M(D') \in S]} \leq e^{\varepsilon}$，这表示算法 M 在两个相邻数据集上输出相同结果的概率比值在$[e^{-\varepsilon}, e^{\varepsilon}]$之间，也就是说，隐私参数ε可以用来控制算法 M 在两个相邻数据集上输出分布的差异，体现了差分隐私的保护水平。$\varepsilon \geq 0$，ε越小表示隐私保护水平越高，相反ε数值越大表示隐私损失越大。当$\varepsilon = 0$ 时隐私保护水平达到最高，意味着对于任意相邻数据集，算法将输出两个概率分布完全相同的结果，这样的结果不能揭示任何关于数据集的信息。一般来讲，ε在小于 1 的情况下能提供比较高的隐私保障。

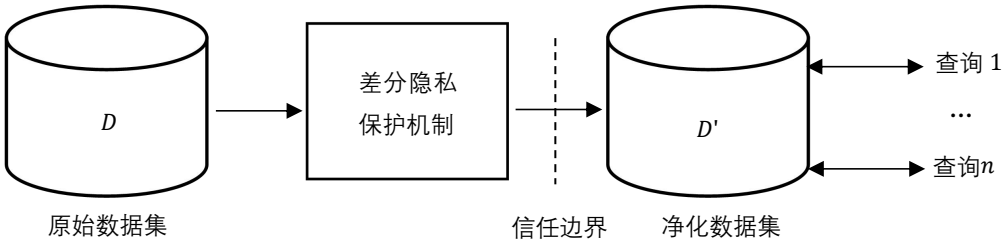
从另一方面来看，ε的取值也反映了数据的可用性，在普通情况下，ε越小，数据可用性越低。ε越大，隐私保护越弱，但数据可用性越高。在一些场景中，必须在ε取值较大的情况下，才能实现有效的数据分析或模型训练任务。因此，隐私预算ε的取值需要结合实际场景和需求，在输出结果的隐私性和可用性之间进行权衡。

2. 差分隐私模型

根据对于多次查询的响应方法不同，差分隐私的发布模型分为交互式和非交互式两种。



（a）交互式数据发布方案



（b）非交互式数据发布方案

在交互式数据发布中，给定数据集 D 和查询集 $F = \{f_1, ..., f_m\}$，需通过一种数据发布机制，使其能够在满足差分隐私保护的条件下，逐个回答查询集　中的查询，直到耗尽全部隐私预算。发布机制的性能通常由精确度来衡量。交互式数据发布即是要在满足一定精确度的条件下，以给定的隐私保护预算回答尽可能多的查询。交互式设置的方法主要考虑事务数据库、直方图、流数据和图数据发布等。

在非交互数据发布中，给定数据集 D 和查询集 $F = \{f_1, ..., f_m\}$，需通过一种数据发布机制，使其能够在满足差分隐私保护的条件下，一次性回答 F 中的所有查询。数据管理者针对所有可能的查询，在满足差分隐私的条件下一次性发布所有查询的结果，或者发布一个原始数据集的净化版本，即带噪音的合成数据集，用户可对合成数据集自行进行所需的查询操作。非交互式数据发布方法主要集中在批查询、列联表发布、基于分组的发布方法以及净化数据集发布。相比于交互式数据发布场景每次查询都要消耗隐私预算，非交互式只需在发布合成数据集时消耗隐私，由于差分隐私的后处理不变性，对合成数据集的后续查询任务，不会进一步泄露原始数据集的隐私。

3. 拉普拉斯机制

差分隐私引入了灵敏度的概念，其度量当数据集有一条改动的情况下，对查询结果的最大改变。

全局灵敏度定义： 设有查询函数 $f: X^n \to R^d$，输入为一个数据集，输出为 d 维实数向量。对于任意相邻数据集 D 和 $D' \in X^n$，则

$$GS_f = \max_{D,D'} ||f(D) - f(D')||_p$$

称为函数f的全局灵敏度。其中$||f(D) - f(D')||_p$是 $f(D)$和 $f(D')$之间的 $p-$阶范数距离，记为l_p灵敏度。对于不同的机制，灵敏度的范数也不同，拉普拉斯机制使用 1 阶范数距离（即曼哈顿距离），而高斯机制采用二阶范数（即欧几里得距离），具体取决于对应机制的隐私分析方法。

4. 拉普拉斯分布

位置参数为 0 的拉普拉斯分布的概率密度函数如图 6-3 所示，其中横轴表示随机变量 x 的取值，纵轴表示相对应的概率密度，b 是拉普拉斯分布的尺度参数。

我们可以看到，拉普拉斯分布的特点是当 $x = 0$ 时，概率密度最大；而在两侧，其概率密度呈指数型下降，特别地，在分布任意一侧中，相同间隔的概率密度比值均相同。

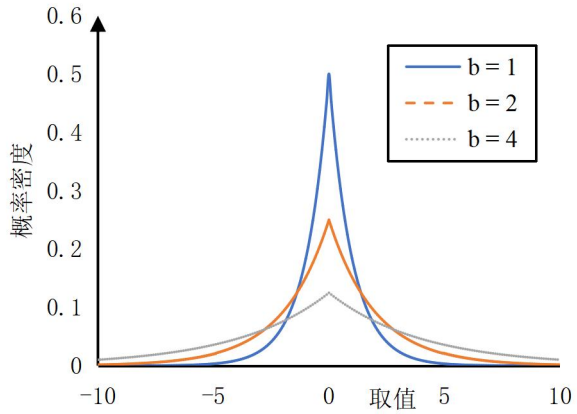


图 6-3 拉普拉斯分布概率密度函数

5. 拉普拉斯机制

拉普拉斯机制定理： 设函数 $f: X^n \to R^k$，$\Delta f = \max_{D,D'} ||f(D) - f(D')||_1$为关于函数 f 的$l_1$灵敏度。给定数据集 $D \in X^n$，则随机算法

$$M(D) = f(D) + (Y_1, \cdots, Y_k), Y_i \sim Lap(\Delta f/\varepsilon)$$

提供ε −差分隐私。其中Y_i是独立同分布的，服从尺度参数 b 为$\Delta f/\varepsilon$的拉普拉斯分布的随机变量。

从不同参数的拉普拉斯分布还可以看出，在灵敏度不变时，ε越小，引入的噪声越大。

6. 拉普拉斯噪声生成

为了实现拉普拉斯机制的加噪过程，首先，我们需要能够产生服从于拉普拉斯分布的噪音。具体来说，我们利用了产生随机变量的组合方法来产生这个噪音。该定理可以描述如下：

若随机变量ζ服从于离散分布$\{p_i\}$, 即 $P(\zeta = i) = p_i$, 同时有 z 服从于$F_\zeta(x)$，取 $z = x$，则有

$$z \sim F(x) = \sum_{i=1}^{K} p_i F_i(x)$$

根据该定理，我们可以得到一个产生符合特定分布的随机数的组合算法：

（1）产生一个正随机数ζ，使得$P(\zeta = i) = p_i, (i = 1,2,3\ldots, K)$；

（2）在$\zeta = i$时，产生具有分布函数$F_i(x)$的随机变量 x。

该算法首先以概率p_i来选择一个子分布函数$F_i(x)$，然后取$F_i(x)$的随机数来作为$F(x)$的随机数。

而具体到拉普拉斯机制而言，如前文所述，由于其概率密度分布函数为：

$$f(x/\beta) = \frac{1}{2\beta} e^{-\frac{|x|}{\beta}}$$

其均值为 0，方差为$2\beta^2$。基于前述的组合算法，我们得到产生拉普拉斯随机数的方法如下：

（1）首先，产生均匀分布的随机数u_1和u_2, 即$u_1, u_2 \sim U(0,1)$；

（2）计算 $x = \begin{cases} \beta \ln(2u_1) + u_2 & u_1 \leqslant 0.5 \\ u_2 - \beta \ln(2(1-u_1)) & u_1 > 0.5 \end{cases}$。

其中，x/β为拉普拉斯机制的隐私参数。

三、实验过程

1. uniform_data 利用混合同余法产生（a, b）区间上均匀分布的随机数

```
/*
函数功能:   利用混合同余法产生（a,b）区间上均匀分布的随机数
输入参数说明:
a       给定区间的下限
b       给定区间的上限
seed    长整型指针变量,  *seed 为伪随机数的种子
*/
double uniform_data(double a, double b,long int * seed)
{
    double t;
    *seed = 2045.0 * (*seed) + 1;
    *seed = *seed - (*seed / 1048576) * 1048576;
    t = (*seed) / 1048576.0;
    t = a + (b - a) * t;
    return t;
}
```

函数 uniform_data 给出了一个生成服从均匀分布 U(0,1)的随机数的算法。该算法利用从主函数中获取的随机数种子 seed 和区间上界 a 及下界 b 作为输入。首先，算法利用线性同余法来对随机种子进行处理，从而生成一个随机数 t。线性同余法的公式为$X_{n+1} = aX_n + c \bmod m$.此处设定 $a = 2045, m = 1048576, c = 1$。其中，2045 和 1048576 是线性同余法中使用的常数，选择它们是为了产生高质量的伪随机数序列。具体来说，2045 是一个较大的质数，它可

以确保生成的随机数序列具有较长的周期，即生成的随机数序列不会很快地重复。进一步地，我们将 t 映射到区间(a,b)上，就完成了一个服从 U(a,b)的随机数生成。

2. 求解 laplace 分布概率累积的反函数并产生 laplace 分布的随机数

```c
/*
函数功能:   求解laplace分布概率累积的反函数，利用该反函数产生laplace分布的随机数
输入参数说明:
beta        拉普拉斯分布参数
seed        长整型指针变量， *seed 为伪随机数的种子
*/
double laplace_data(double beta, long int * seed)
{
    double u1,u2, x;
    u1 = uniform_data(0.0, 1.0, seed);
    u2 = uniform_data(0.0, 1.0, seed);
    if (u1 < 0.5)
    {
        x = beta * (log(2*u1)+u2);
    }
    else
    {
        x = u2 - (beta * log(2*(1-u1)));
    }

    return x;
}
```

函数 laplace_data 以隐私预算 β 和随机数种子 seed 作为输入，生成一个服从拉普拉斯分布的随机数。该算法首先调用 uniform_data 函数产生两个服从 U(0,1)的随机数。将这两个随机数代入到此前的公式，这样，就可以得到一个以隐私参数 β 和随机种子 seed 生成的服从拉普拉斯分布的随机数了。

3. 提取 csv 文件并生成拉普拉斯分布的噪音对查询返回的结果进行加噪

非交互式发布不在结果上加噪音，而在数据上加噪音，产生加噪后的数据，以产生的加噪的数据集来响应查询。在这种情况下，需要对每条记录都根据设定的隐私预算产生并添加相关的拉普拉斯噪声，进而生成合成的数据集。

非交互式发布需要对数据集中的数据添加噪音,而交互式发布需要对查询返回的结果添加噪音。

```c
/*
函数功能:   对传入的csv文件进行处理，提取其中数据并生成拉普拉斯分布的噪音进行加噪
输入参数说明:
path        csv文件的存储位置
beta        拉普拉斯分布参数
seed        长整型指针变量， *seed 为伪随机数的种子
*/
void csv_analysis(char* path, double beta, long int seed)
{
    FILE *original_file = fopen(path,"r+"); //读取指定路径的数据集
    struct Animals * original_data = NULL;
    original_data = hb_csv_parser(original_file);
    int sum=0,i=0;
    double x = 0;
    while(original_data[i].name)  //循环为原始数据集内各条数据生成拉普拉斯噪音并加噪
    {
        x = laplace_data(beta,&seed); //产生拉普拉斯随机数
        printf("Added noise:%f\t%s\t%f\n",x,original_data[i].name,original_data[i].carrots+x); //此处分别列出了每条具体添加的噪音和加噪的结果。当投入较少预算时，可能会出现负数
        if (original_data[i].carrots + x >= 55)
        {
            sum++;
        }
        i++;
    }
    printf("Animals which carrots cost > 55 (Under DP): %d\n",sum); //输出加噪后的数据集中，每日食用胡萝卜大于55的动物个数
}
```

在 csv_analysis 函数完成对 csv 数据集的读取后，会进入循环体，直接在结果上添加 laplace 噪声，而不是对数据集添加噪声。

4. 交互式发布主函数

主函数中，指定全局敏感度为 1，指定隐私预算为 0.1，支持查询次数为 20 次，并生成基于时间的随机种子。

设计一个相邻数据集 zoo_nb.csv（去掉了“Dugong”这一项数据），来进行对比演示加入不同规模的噪音对统计结果的影响。利用这两个隐私预算分别基于原始数据集和相邻数据集对查询返回的结果进行加噪并进行前述查询，以此来对噪声的影响进行展示和比较。

```
long int seed;
int sen = 1;  //对于一个单属性的数据集，其敏感度为1
double beta = 0;
srand((unsigned)time(NULL)); //生成基于时间的随机种子（srand方法）
printf("Please input laplace epsilon:");
scanf("%lf", &beta);
if (beta <= 0 || !beta)//当输入的beta值无效时，默认设定beta值为1
{
    beta = 1.0;
}
int search_times;
printf("Please input search times:");
scanf("%ld", &search_times);

printf("Under privacy budget %f, sanitized original data with fake animal name and laplace noise:\n", beta);
printf("Every single search privacy budget %f\n", beta / 20);
beta = sen / (beta / 20); //拉普拉斯机制下，实际公式的算子beta为敏感度/预算
double avg_old = 0;
for (int i = 0; i < search_times; i++)
{
    seed = rand() % 10000 + 10000; //随机种子产生
    avg_old += csv_analysis("./zoo.csv", beta, seed); //先调用原始数据集
}
printf("Avg Old Search Result: \t%f\n", avg_old / search_times);
printf("================Using neighbour dataset================\n");
double avg_new = 0;
for (int i = 0; i < search_times; i++)
{
    seed = rand() % 10000 + 10000; //随机种子更新
    avg_new += csv_analysis("./zoo_nb.csv", beta, seed); //再调用相邻数据集
}
printf("Avg New Search Result: \t%f\n", avg_new / search_times);
return 0;
```

四、实验结果

在编译的时候，可以看到如下报错

```
gcc csvpackage.o laplace.o testraw.o -o testraw  -lm
/usr/local/bin/ld: testraw.o:(.bss+0x0): multiple definition of `Ani'; csvpackage.o:(.bss+0x0): first defined here
/usr/local/bin/ld: testraw.o:(.bss+0x10): multiple definition of `Hb'; csvpackage.o:(.bss+0x10): first defined here
collect2: error: ld returned 1 exit status
make: *** [Makefile:7：testraw] 错误 1
```

在编译的时候，可以看到如下报错
在./include/csvpackage.h 可以看到

```c
/*
结构体：     Animals
为zoo数据集提供结构化存储功能
成员说明：
name        对应zoo数据集的第一列，动物名称
carrots     对应zoo数据集的第二列，每日食用胡萝卜数量
*/
struct Animals{
    char* name;
    int carrots;
}Ani;

/*
结构体：     Histobuckets
为medicaldata数据集提供结构化存储功能
成员说明：
bucket      对应medicaldata数据集的第一列，分桶名称
count       对应medicaldata数据集的第二列，桶内元素数量
*/
struct Histobuckets{
    char* bucket;
    int count;
}Hb;
```

需要把 Ani 和 Hb 实例删除

```c
/*
结构体：     Animals
为zoo数据集提供结构化存储功能
成员说明：
name        对应zoo数据集的第一列，动物名称
carrots     对应zoo数据集的第二列，每日食用胡萝卜数量
*/
struct Animals {
    char* name;
    int carrots;
};

/*
结构体：     Histobuckets
为medicaldata数据集提供结构化存储功能
成员说明：
bucket      对应medicaldata数据集的第一列，分桶名称
count       对应medicaldata数据集的第二列，桶内元素数量
*/
struct Histobuckets {
    char* bucket;
    int count;
};
```

输入 0.1 和 20 后，观察程序运行结果

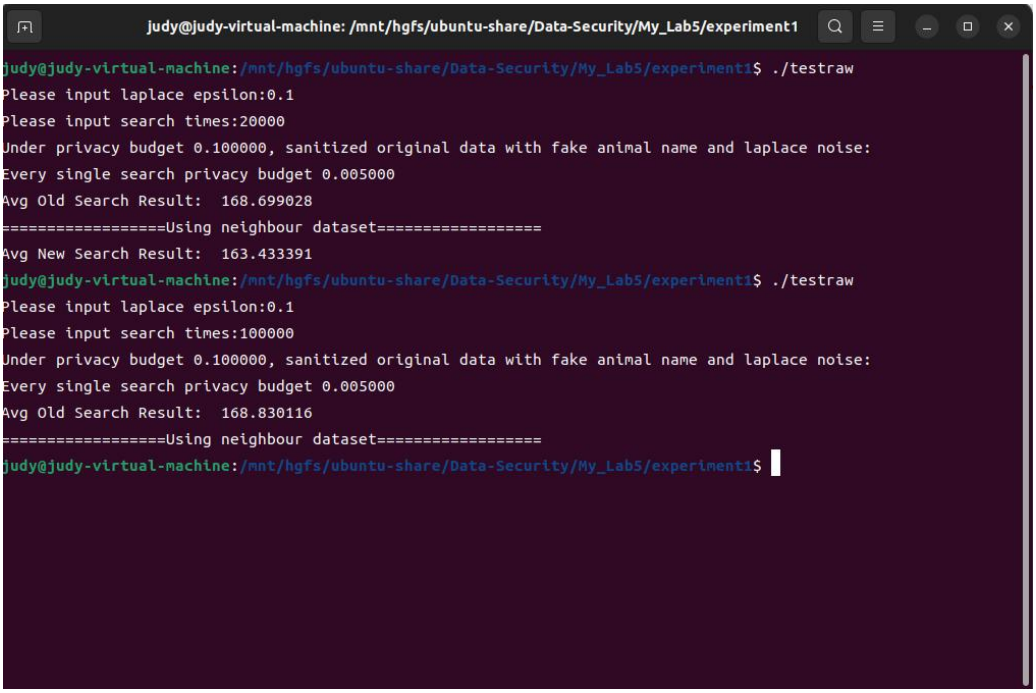
```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab5/experiment1$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original data with fake animal name and laplace noise:
Every single search privacy budget 0.005000
Animals which carrots cost > 55 (original): 90
Added noise:-371.450944 Aardvark        -370.450944
Added noise:96.512365   Albatross       184.512365
Added noise:-111.926824 Alligator       -76.926824
Added noise:275.640499  Alpaca  374.640499
Added noise:71.223465   Ant     140.223465
Added noise:53.904789   Anteater        67.904789
Added noise:82.336821   Antelope        159.336821
Added noise:-18.096444  Ape     34.903556
Added noise:-722.751933 Armadillo       -628.751933
Added noise:50.826146   Baboon  117.826146
Added noise:-133.372073 Badger  -41.372073
```

```
Added noise:74.342284   Whale   161.342284
Added noise:3.491039    Wolf    84.491039
Added noise:-34.384560  Wolverine       1.615440
Added noise:1060.507162 Wombat  1144.507162
Added noise:106.999953  Woodcock       160.999953
Added noise:363.274880  Woodpecker      370.274880
Added noise:-8.048426   Worm    33.951574
Added noise:800.926186  Wren    855.926186
Added noise:-222.208207 Yak     -162.208207
Added noise:204.096008  Zebra   211.096008
Animals which carrots cost > 55 (Under DP): 114
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab5/experiment1$
```

查看平均值，可以看到此时抗攻击效果很好。

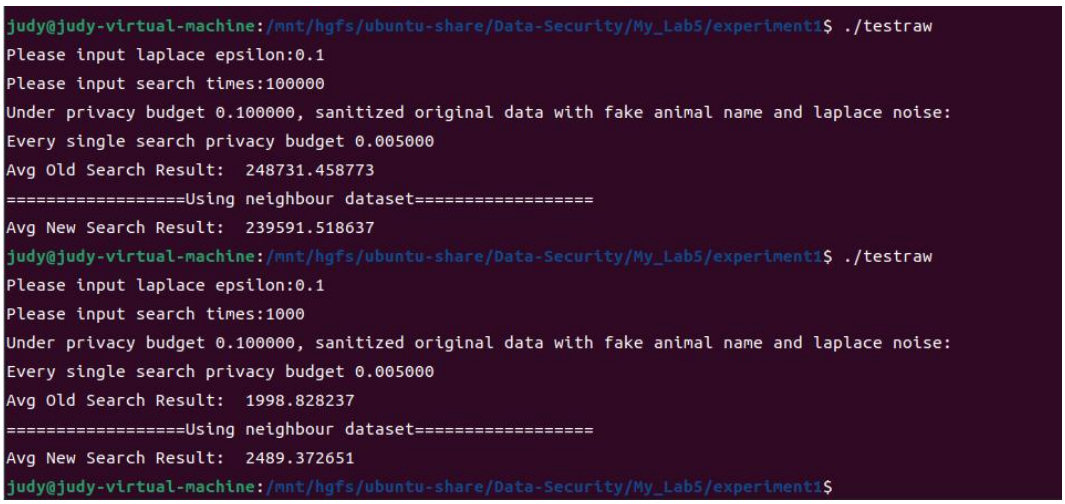
```
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab5/experiment1$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original data with fake animal name and laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result:  135.530035
==================Using neighbour dataset==================
Avg New Search Result:  102.234824
```

下面尝试 epsilon=0.1 ，增加 search_times,如果隐私预算被耗尽的情形



可以看到平均值开始逐渐接近，落在 160-170 的区间内，可以认定隐私泄露。

而如果把算子 beta 改为 beta=sen/(beta/search_times)将不会出现隐私泄露的情况，可以看到平均值不在一个区间内，可以认为隐私信息未发生泄露



五、心得体会

本次实验主要是针对交互式发布的差分隐私方案进行评估，并通过实验验证隐私保护的效果。在实验中，我们首先了解了差分隐私的基本原理，即通过向查询结果添加一定的噪音来保护隐私。然后，介绍了拉普拉斯机制作为一种常用的差分隐私发布机制，以及如何生成符合拉普拉斯分布的噪音。

在实验过程中，我们实现了生成均匀分布和拉普拉斯分布的随机数的算法，并且对查询结果进行了加噪处理。针对非交互式发布，我们在数据集上添加噪音，

而对于交互式发布，则在查询结果上添加噪音。通过比较加入不同规模噪音的查询结果，展示了噪音对统计结果的影响。

在实验结果分析中，我们观察到在设定较小的隐私预算下，对查询结果的影响较小，隐私保护效果良好。但是当隐私预算被耗尽时，查询结果开始逐渐接近原始数据的情况下，隐私泄露可能发生。此时，调整算子参数或增加搜索次数可以有效避免隐私泄露，使得查询结果保持较大的波动性，从而保证了隐私的安全性。

通过本次实验，我深入了解了差分隐私的概念和原理，并掌握了一种常见的差分隐私发布机制。同时，实践中也发现了在设计隐私方案时需要综合考虑隐私预算、查询次数等因素，并通过调整参数来平衡隐私保护和数据可用性之间的关系。这对于今后在数据安全领域的研究和实践都具有指导意义。