



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

数据安全

交互式发布 DP 方案评估

穆禹宸 2012026

年级：2020 级

专业：信息安全、法学双学位班

指导教师：刘哲理

2023 年 6 月 1 日

目录

一、 实验名称	1
二、 实验要求	1
三、 实验过程	1
(一) 实验原理	1
(二) 实验代码	4
(三) 实验结果	5
四、 心得体会	9
五、 附录：完整代码	10

一、 实验名称

交互式发布 DP 方案评估

二、 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 发布后的结果进行评估说明隐私保护的效果。

三、 实验过程

(一) 实验原理

首先, 我们需要了解查询灵敏度这个概念: 数量查询的灵敏度为 1 根据定义, $\Delta f = \max_{D, D'} \|f(D) - f(D')\|$, 有随机算法 $M(D) = f(D) + Y, Y \sim \text{Lap}(1/\epsilon)$, 能够提供 ϵ -差分隐私保护。我们在代码中有如下定义:

```
1 int sen = 1; // 对于一个单属性的数据集, 其敏感度为 1
```

除此之外, 如果我们仅把 ϵ 本身作为生成拉普拉斯噪音的参数, 由于 Laplace 噪音的数学期望为 0, 在多次查询后取平均值就可以在在一定程度上暴力推断出隐私信息。因此, 对于我们交互式的方案, 要考虑保护多次查询的话, 需要为每次查询进行预算分配: 假定隐私预算为 ϵ , 允许的查询次数为 k , 则每次查询分配的预算为 ϵ/k , 这样才能达到 ϵ -差分隐私的目标。注意, 在交互式发布 DP 方案之中, 每一次查询其实都会造成对应的隐私损失。因此, 在数学上每次查询添加的噪声均服从 $\text{Lap}(0, k/\epsilon)$ 分布。

```

1 // 头文件之中
2 /*
3 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
4 输入参数说明:
5 beta 求解 laplace 分布参数
6 seed 长整型指针变量, *seed 为伪随机数的种子
7 */
8 double laplace_data(double beta, long int * seed)
9 {
10  double u1,u2, x;
11  double u1 = uniform_data(0.0, 1.0, seed);
12  double u2 = uniform_data(0.0, 1.0, seed);
13  if (u1 < 0.5)
14  {
15      x = beta * (log(2*u1)+u2);
16  }
17  else
18  {
19      x = u2 - (beta * log(2*(1-u1)));
20  }
21  return x;
22 }
23 // 主函数之中
24 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

```

ϵ	噪声绝对值的数据分布				多次查询添加噪声的平均值落在危险区间内的概率			
	90%	95%	99%	99.9%	100	1000	10000	100000
1	2.29	2.98	4.50	6.43	100.00	100.00	100.00	100.00
0.1	23.24	29.99	45.51	66.56	25.59	73.75	99.99	100.00
0.01	227.97	296.22	463.48	677.26	2.72	9.12	27.85	73.70

图 1: 多次查询

注意上图, 就展示了多次查询后被推测出的概率。

同时, 在隐私保护中, 随机数种子的选择非常重要。在上面的代码之中, 它符合这样的数学公式:

$$x = \begin{cases} \beta \ln(2u_1) + u_2, & u_1 \leq 0.5; \\ u_2 - \beta \ln(2(1 - u_1)), & u_1 > 0.5; \end{cases} \quad (1)$$

这就是本次实验所需要掌握的一定的原理。

然后, 让我们结合原始代码先看一下非交互式的方案:

```

[nyc@ubuntu22][20:25:35]-[~/Desktop/experiment1]
$ ./testraw
Please input laplace epsilon:10
Under privacy budget 10.000000, sanitized original data with fake animal name and laplace noise:
Animals which carrots cost > 55 (original): 90
Added noise:-0.212316 Animal 1 1.212316
Added noise:-0.011108 Animal 2 87.988892
Added noise:0.024492 Animal 3 35.024492
Added noise:-0.116562 Animal 4 98.883438
Added noise:0.643269 Animal 5 69.643269
Added noise:0.035929 Animal 6 14.035929
Added noise:0.595973 Animal 7 77.595973
Added noise:0.492493 Animal 8 53.492493
Added noise:0.033846 Animal 9 94.033846
Added noise:-0.099334 Animal 10 66.900666
Added noise:0.222261 Animal 11 92.222261
Added noise:0.608809 Animal 12 87.608809
Added noise:0.750655 Animal 13 70.750655
Added noise:-0.015943 Animal 14 30.984057

```

图 2: 原始方案

先把 ϵ 设置为 10 观察一下结果

```

Added noise:-0.340696 Animal 180 54.659304
Added noise:-0.015843 Animal 181 59.984157
Added noise:0.968488 Animal 182 7.968488
Animals which carrots cost > 55 (Under DP): 88
=====Using neighbour dataset=====
Animals which carrots cost > 55 (original): 89
Added noise:-0.025436 Animal 1 0.974564
Added noise:0.222147 Animal 2 88.222147
Added noise:0.223020 Animal 3 35.223020
Added noise:1.101219 Animal 4 100.101219
Added noise:-0.061268 Animal 5 68.938732

```

图 3: 原始方案

```

Added noise:0.375213 Animal 173 81.375213
Added noise:0.235221 Animal 174 36.235221
Added noise:-0.053679 Animal 175 83.946321
Added noise:-0.259880 Animal 176 53.740120
Added noise:-0.054230 Animal 177 6.945770
Added noise:1.122268 Animal 178 43.122268
Added noise:0.030331 Animal 179 55.030331
Added noise:0.456430 Animal 180 60.456430
Added noise:0.228606 Animal 181 7.228606
Animals which carrots cost > 55 (Under DP): 89

```

图 4: 原始方案

ϵ 越大可用性越好，我们可以观察到这个时候的噪声非常小。

```

[nyc@ubuntu22][20:26:41]-[~/Desktop/experiment1]
$ ./testraw
Please input laplace epsilon:0.1
Under privacy budget 0.100000, sanitized original data with fake animal name and laplace noise:
Animals which carrots cost > 55 (original): 90
Added noise:-0.213359 Animal 1 0.786641
Added noise:0.940410 Animal 2 88.940410
Added noise:8.719973 Animal 3 43.719973
Added noise:-4.251401 Animal 4 94.748599
Added noise:17.940259 Animal 5 86.940259
Added noise:5.878423 Animal 6 19.878423
Added noise:28.709010 Animal 7 105.709010
Added noise:-9.415673 Animal 8 43.584327
Added noise:8.502647 Animal 9 102.502647
Added noise:-10.644010 Animal 10 56.355990
Added noise:2.313694 Animal 11 94.313694
Added noise:24.801765 Animal 12 111.801765
Added noise:-4.462172 Animal 13 65.537828
Added noise:17.735170 Animal 14 48.735170
Added noise:-9.309144 Animal 15 4.690856
Added noise:-11.623875 Animal 16 2.376125
Added noise:1.288727 Animal 17 62.288727
Added noise:-35.529468 Animal 18 21.470532
Added noise:7.681205 Animal 19 75.681205
Added noise:1.384066 Animal 20 14.384066
Added noise:18.176829 Animal 21 39.176829
Added noise:-1.582825 Animal 22 36.417175
Added noise:-14.367428 Animal 23 77.632572
Added noise:-4.510904 Animal 24 34.483096
Added noise:27.492390 Animal 25 73.492390

```

图 5: 原始方案

```

Added noise:-27.113072 Animal 180 37.113072
Added noise:-15.727457 Animal 181 44.272543
Added noise:6.763122 Animal 182 13.763122
Animals which carrots cost > 55 (Under DP): 95
=====Using neighbour dataset=====
Animals which carrots cost > 55 (original): 89
Added noise:3.635208 Animal 1 4.635208
Added noise:7.406737 Animal 2 95.406737
Added noise:6.996034 Animal 3 41.996034
Added noise:-22.516742 Animal 4 76.483258

```

图 6: 原始方案

从统计到的查询结果来看，变化也并不明显。这说明这时候数据的隐私保护效果一般。

```

Added noise:15.984599 Animal 171 35.984599
Added noise:-2.602610 Animal 172 84.397390
Added noise:12.599322 Animal 173 93.599322
Added noise:6.171673 Animal 174 42.171673
Added noise:-3.441509 Animal 175 80.558491
Added noise:1.859005 Animal 176 55.859005
Added noise:-1.177248 Animal 177 5.822752
Added noise:-8.855792 Animal 178 33.144208
Added noise:1.616296 Animal 179 56.616296
Added noise:15.933890 Animal 180 75.933890
Added noise:4.809637 Animal 181 11.809637
Animals which carrots cost > 55 (Under DP): 94

```

图 7: 原始方案

而后续我们将会直接进行 $\epsilon = 0.1$ 的实验结果，让我们后续观察效果。

(二) 实验代码

本次实验主要进行复现和修改代码，观察结果并进行现象分析，**以下是我的修改和理解**：本次实验就是要理解交互式和非交互式的区别：

```

1 while(original_data[i].name) //循环为原始数据集内各条数据去除标识（动物名）、生成拉
  ↳ 普拉斯噪音并加噪
2 {
3     x = laplace_data(beta,&seed); //产生拉普拉斯随机数
4     //printf("Added noise:%f\t%s",x,"Animal",i+1,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
5     if(original_data[i].carrots+x>=55)
6     {
7         sum++;
8     }
9     i++;
10 }
11 printf("Animals which carrots cost > 55 (Under DP): %d\n",sum+(int)x); //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
12 return sum+x;

```

注意上面的函数，我们是直接在结果上添加了 Laplace 噪声，而不是对原来的数据集进行噪声添加的工作了。

然后我们在主函数之中进行查询次数的设置：

```

1 int search_times;
2 printf("Please input search times:");
3 scanf("%ld", &search_times);

```

当然，我也对多次查询取平均值的结果做了一定的统计工作：

```

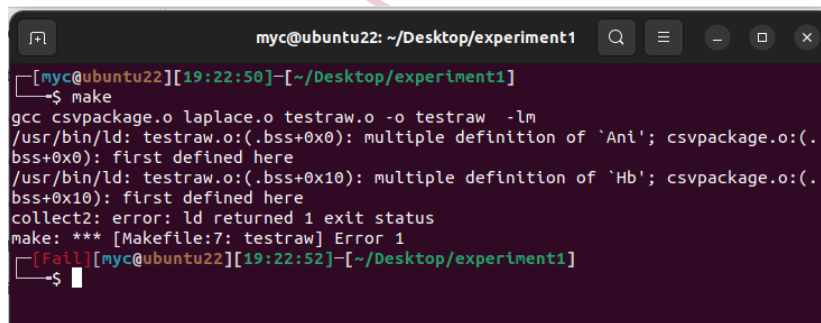
1 double avg_old=0;
2 for(int i=0;i<search_times;i++){
3     seed = rand()%10000+10000; //随机种子产生
4     avg_old+=csv_analysis("./zoo.csv",beta,seed); //先调用原始数据集
5 }
6 printf("Avg Old Search Result: \t%f\n", avg_old/search_times);
7 printf("=====Using neighbour dataset=====\n");
8 double avg_new=0;
9 for(int i=0;i<search_times;i++){
10    seed = rand()%10000+10000; //随机种子更新
11    avg_new+=csv_analysis("./zoo_nb.csv",beta,seed); //再调用相邻数据集
12 }
13 printf("Avg New Search Result: \t%f\n", avg_new/search_times);

```

以上是全部修改的代码，我们将在后面结合实验结果进行更多针对性的介绍。

(三) 实验结果

事实上，实验所提供的参考代码在 ubuntu22 以及最新版本的 gcc 的环境下并不能够顺利编译通过，如下图所示：



```

myc@ubuntu22: ~/Desktop/experiment1
$ make
gcc csvpackage.o laplace.o testraw.o -o testraw -lm
/usr/bin/ld: testraw.o(.bss+0x0): multiple definition of `Ani'; csvpackage.o(.bss+0x0): first defined here
/usr/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] Error 1
[Fail] myc@ubuntu22 [19:22:52] ~/Desktop/experiment1
$

```

图 8: 遇到问题

这是由于重定义问题，我们删除 csvpackage.h 中的 Ani 和 Hb 实例，编译通过。

```
myc@ubuntu22: ~/Desktop/experiment1
[myc@ubuntu22][19:25:22]-[~/Desktop/experiment1]
$ make
gcc -I./include -c csvpackage.c
gcc -I./include -c laplace.c -lm
gcc -I./include -c testraw.c
gcc csvpackage.o laplace.o testraw.o -o testraw -lm
gcc -I./include -c testhist.c
gcc csvpackage.o laplace.o testhist.o -o testhist -lm
[myc@ubuntu22][19:25:24]-[~/Desktop/experiment1]
$
```

图 9: 顺利解决

然后才能正常开始实验。

```
[myc@ubuntu22][21:50:48]-[~/Desktop/experiment1]
$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original data with fake and
d laplace noise:
Every single search privacy budget 0.005000
Animals which carrots cost > 55 (original): 90
Added noise:-69.791679 Animal 1 -68.791679
Added noise:757.161959 Animal 2 845.161959
Added noise:193.834090 Animal 3 228.834090
Added noise:2.069390 Animal 4 101.069390
Added noise:13.971449 Animal 5 82.971449
Added noise:-271.691835 Animal 6 -257.691835
Added noise:-18.947005 Animal 7 58.052995
Added noise:382.396686 Animal 8 435.396686
Added noise:64.132284 Animal 9 158.132284
Added noise:87.975354 Animal 10 154.975354
Added noise:108.584968 Animal 11 200.584968
Added noise:161.872847 Animal 12 248.872847
Added noise:64.548578 Animal 13 134.548578
Added noise:28.671548 Animal 14 59.671548
Added noise:14.546549 Animal 15 28.546549
Added noise:182.212720 Animal 16 196.212720
Added noise:-112.067488 Animal 17 -51.067488
Added noise:124.302683 Animal 18 181.302683
Added noise:152.337011 Animal 19 220.337011
Added noise:25.599481 Animal 20 38.599481
Added noise:88.763514 Animal 21 109.763514
Added noise:-261.866693 Animal 22 -223.866693
Added noise:-39.087112 Animal 23 52.912888
Added noise:-286.366425 Animal 24 -247.366425
Added noise:15.353758 Animal 25 61.353758
Added noise:-424.712804 Animal 26 -388.712804
Added noise:120.558815 Animal 27 143.558815
Added noise:25.655343 Animal 28 101.655343
```

图 10: 开始实验

输入 0.1 和 20 后，观察程序的运行结果：

```
[myc@ubuntu22][21:54:58]-[~/Desktop/experiment1]
$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original
d laplace noise:
Every single search privacy budget 0.005000
Animals which carrots cost > 55 (original): 90
Animals which carrots cost > 55 (Under DP): 311
Animals which carrots cost > 55 (original): 90
```

图 11: 部分结果

这时候可以非常明显的看到隐私保护的效果，原始结果和加入噪声后的结果有着很大差异。


```

Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 324
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 127
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 92
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 32
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 55
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 42
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 302
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): 235
Animals which carrots cost > 55 (original): 89
Animals which carrots cost > 55 (Under DP): -497

```

图 12: 部分结果

当然, 当 $\epsilon = 0.1$, $k = 20$ 时, 拉普拉斯分布的方差非常大, 那么这时候噪声会特别大, 因此数据的可用性也很差!

然后我们看一下平均值:

```

[myc@ubuntu22][21:57:03]-[~/Desktop/experiment1]
--$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original data with fa
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 146.525092
=====Using neighbour dataset=====
Avg New Search Result: 171.906165
[myc@ubuntu22][21:57:09]-[~/Desktop/experiment1]
--$

```

图 13: 平均值

可以看出有着巨大不同, 此时抗攻击效果很好。也就是说, 即使取平均进行重复攻击, 数据也被很好地保护了, 没有泄露隐私信息。

```

[myc@ubuntu22][21:57:03]-[~/Desktop/experiment1]
--$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20
Under privacy budget 0.100000, sanitized original data with fa
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 146.525092
=====Using neighbour dataset=====
Avg New Search Result: 171.906165
[myc@ubuntu22][21:57:09]-[~/Desktop/experiment1]
--$

```

图 14: 平均值

然后, 我又做了更多实验内容, 尝试了一下当隐私预算被耗尽会怎么样。

```

[myc@ubuntu22][22:02:07]-[~/Desktop/experiment1]
--$ ./testraw
Please input laplace epsilon:0.1
Please input search times:100
Under privacy budget 0.100000, sanitized original data wi
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 208.098049
=====Using neighbour dataset=====
Avg New Search Result: 154.622675
[myc@ubuntu22][22:02:13]-[~/Desktop/experiment1]
--$

```

图 15: 100 次

这时我们已经开始观察到平均值开始接近了。

```
[myc@ubuntu22][22:02:13]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:1000
Under privacy budget 0.100000, sanitized original data
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 173.234307
=====Using neighbour dataset=====
Avg New Search Result: 152.623136
[myc@ubuntu22][22:02:26]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:5000
Under privacy budget 0.100000, sanitized original data
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 166.121353
=====Using neighbour dataset=====
Avg New Search Result: 163.184803
```

图 16: 更多次

```
[myc@ubuntu22][22:02:39]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:10000
Under privacy budget 0.100000, sanitized original da
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 170.124113
=====Using neighbour dataset=====
Avg New Search Result: 166.944502
[myc@ubuntu22][22:02:58]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:20000
Under privacy budget 0.100000, sanitized original da
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 171.403037
=====Using neighbour dataset=====
Avg New Search Result: 163.198714
[myc@ubuntu22][22:03:21]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:8000
Under privacy budget 0.100000, sanitized original da
d laplace noise:
Every single search privacy budget 0.005000
Avg Old Search Result: 169.799946
=====Using neighbour dataset=====
Avg New Search Result: 166.168161
```

图 17: 更多次

最后我们可以看到,当大量重复查询后,隐私预算被消耗殆尽,此时平均值会落在一个 160-170 的区间之内,这就是我们之前提到过的危险区间!

这时我们已经可以认定隐私泄露了!

当然,如果我们隐私预算设置得非常充足(达到理论最好的值),那么就不会有这个问题。首先修改代码:

```
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
```

图 18: 实际预算

```
printf("Every single search privacy budget %f\n", beta/search_times);
beta = sen / (beta/search_times); //拉普拉斯机制下, 实际公式的算子beta
为敏感度/预算
```

图 19: 理论最优预算

这时, 我们再进行多次查询, 会得到如下结果:

```
[myc@ubuntu22][22:09:01]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:100
Under privacy budget 0.100000, sanitized original data with f
d laplace noise:
Every single search privacy budget 0.001000
Avg Old Search Result: 164.282360
=====Using neighbour dataset=====
Avg New Search Result: 509.652990
[myc@ubuntu22][22:09:06]-[~/Desktop/experiment1]
→$ ./testraw
Please input laplace epsilon:0.1
Please input search times:1000
Under privacy budget 0.100000, sanitized original data with f
d laplace noise:
Every single search privacy budget 0.000100
Avg Old Search Result: 2967.507546
=====Using neighbour dataset=====
Avg New Search Result: 2836.152195
```

图 20: 预算充足的查询

这时我们可以明显看到, 平均值根本不会出现在一个区间之内, 那么隐私信息也就不会有任何泄露了。

四、 心得体会

交互式数据库查询的隐私保护必须考虑到隐私泄露的风险。通过添加随机噪音的方式来保护查询的隐私, 这是一个常用且实用的技术。拉普拉斯机制是一种可行的机制, 它通过向查询结果添加自适应噪音来保护数据隐私, 可以有效地保护数据隐私, 同时也具有简单快捷的优点。

此次实验我们使用了一个实现隐私预算的拉普拉斯机制的程序, 该程序通过读取指定路径的数据集, 然后遍历这些数据集中的元素, 将每个元素加上拉普拉斯分布的随机数, 实现对原始数据集的加噪处理。在加噪后的结果中, 每条数据的信息都被拉普拉斯噪声所掩盖, 达到保护数据隐私的目的。

此外, 为了保证查询结果的准确性, 我们需要使用随机生成的种子, 以确保随机性。同时, 需要根据预算和敏感度计算出拉普拉斯机制需要的实际参数, 以提高机制的效率和准确性。

当然, 对于这个方案, 更多的探索则是如何平衡隐私保护和数据可用性之间的关系。进一步讲, 是否存在可行的办法可以计算最优的 ϵ 和 K 值? 在我查阅的一些资料之中, 确实有一些方法可以进行一定条件下的最优参数参数计算, 这一点还有待后续进一步探索。

同时, 本次实验之中我思考过如何展示隐私预算的消耗情况, 遗憾的是没有展示出来, 只能通过噪声的对比进行肉眼观察和展示, 然而并不直观, 因此没有撰写在报告之中。

五、 附录：完整代码

实验完整代码如下所示：

完整代码

```

1  #include <stdio.h>
2  #include <string.h>
3  #include "laplace.h"
4  #include "csvpackage.h"
5  #include <time.h>
6  extern int rand();
7  extern void srand(unsigned);
8  /*
9  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
10  输入参数说明：
11  path          csv文件的存储位置
12  beta          拉普拉斯分布参数
13  seed          长整型指针变量， *seed 为伪随机数的种子
14  */
15  double csv_analysis(char* path, double beta, long int seed)
16  {
17      FILE *original_file = fopen(path,"r+"); //读取指定路径的数据集
18      struct Animals * original_data = NULL;
19      original_data = csv_parser(original_file);
20      int sum=0,i=0;
21      double x = 0;
22      while(original_data[i].name) //循环为原始数据集内各条数据去除标识
          (动物名)、生成拉普拉斯噪音并加噪
23      {
24          x = laplace_data(beta,&seed); //产生拉普拉斯随机数
25          //printf("Added noise:%f\t%s %d\t%f\n",x,"Animal",i+1,
          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
26          if(original_data[i].carrots+x>=55)
27          {
28              sum++;
29          }
30          i++;
31      }
32      //printf("Animals which carrots cost > 55 (Under DP): %d\n",sum+(int)
          x); //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
33      return sum+x;
34  }
35
36  /*
37  参数表：
38  seed          长整型指针变量， *seed为伪随机数的种子

```

```

39  sen                                数据集的敏感度
40  x                                  用于储存拉普拉斯分布噪音的临时变量
41  beta                              隐私预算，在输入后根据公式转换为拉普拉斯分布参数
42  */
43  int main()
44  {
45      long int seed;
46      int sen = 1; // 对于一个单属性的数据集，其敏感度为1
47      double beta;
48      srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand方法）
49      beta = 0;
50      printf("Please input laplace epsilon:");
51      scanf("%lf", &beta);
52      if(beta<=0 || !beta) //当输入的beta值无效时，默认设定beta值为1
53      {
54          beta = 1.0;
55      }
56      int search_times;
57      printf("Please input search times:");
58      scanf("%ld", &search_times);
59
60      printf("Under privacy budget %f, sanitized original data with fake
        animal name and laplace noise:\n",beta);
61      printf("Every single search privacy budget %f\n", beta/20);
62      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
63      double avg_old=0;
64      for(int i=0;i<search_times;i++){
65          seed = rand()%10000+10000; //随机种子产生
66          avg_old+=csv_analysis("./zoo.csv",beta,seed); //先调用原始数
            据集
67      }
68      printf("Avg Old Search Result: \t%f\n", avg_old/search_times);
69      printf("=====Using neighbour dataset=====\\n
        ");
70      double avg_new=0;
71      for(int i=0;i<search_times;i++){
72          seed = rand()%10000+10000; //随机种子更新
73          avg_new+=csv_analysis("./zoo_nb.csv",beta,seed); //再调用相邻
            数据集
74      }
75      printf("Avg New Search Result: \t%f\n", avg_new/search_times);
76      return 0;
77  }

```