# 第五章 差分隐私

学习要求：DP的基本定义；ESA-EAA模型。

课时：4课时。

差分隐私概念最早由Cynthia Dwork74等人于2006年提出，区别于以往的*k*-匿名等隐私保护方案，其主要贡献是给出了对个人隐私泄露的数学定义，可以在最大化查询结果可用性的同时，保证单个用户隐私泄露不超过预先设定的。差分隐私并不是要求保证数据集的整体性的隐私，而是对数据集中的每个个体的隐私提供保护。

## 基本概念

### 差分攻击

我们在前序的章节中已经讨论过各类匿名化处理数据的方法，也对数据的脱敏进行了充分的讨论。然而，这些算法虽然能一定程度上保护隐私，但是在某些特定的攻击方法下，其仍然会暴露个体的隐私，差分攻击就是其中的一种。

我们以一个例子来说明差分攻击是如何获取到个体隐私的。表5.1展示了一个由学生的姓名、性别和成绩三个属性组成的数据表。现在，我们设姓名和性别是公开数据，可以响应任意针对其的查询，而我们打算保护个体成绩的隐私。因此，我们**要求查询服务器只响应对非个体的（如，所有女生）的成绩查询请求**。

表5.1 学生数据库示例

|  |  |  |
| --- | --- | --- |
| **姓名** | **性别** | **成绩** |
| Aisha | F | Fail |
| Benny | M | Pass |
| Erica | F | Fail |
| Fabio | M | Fail |
| Johan | M | Fail |
| Ming | M | Pass |
| Orhan | M | Pass |

此时，我们来考虑以下四种情况：

(1) 查询“Ming”的成绩。显而易见，这是一个对个体成绩的查询，查询服务器不会响应这样的查询。

(2) 查询“有多少学生通过了考试”。这样的查询是符合要求的，查询服务器会忠实响应这个查询。

(3) 查询“有多少女生通过了考试”。看起来该查询与查询2一样，都是针对一组人而不是一个人的查询，所以查询服务器也会忠实地响应这个查询。但是，这个查询是会泄露敏感信息的。因为在该例子中，所有女生都没有通过测试，所以该查询会返回0。由此我们可以推导出，Aisha和Erica的个体成绩均为Fail，其该属性的个体隐私被暴露了。

(4) 如果说情况3还需要依赖数据本身的性质来暴露个体隐私，那么，现在我们将Aisha的成绩属性置为pass。此时，情况3中的隐私泄露不复存在，因为此时情况3的查询会返回1，由于女生数量为2，我们无从知道具体是哪个女生没有通过。

在这种情况下，我们可以利用另一种方式来达到获取个体隐私的目的。对于个体Aisha，我们首先查询“所有学生中通过考试的人数”，再查询“除了Aisha外所有学生通过考试的人数”。以上两个查询均符合不对个体查询的要求，所以都会被查询服务器执行。此时，第一个问题的结果是4，第二个问题的结果是3，我们就可以推导出Aisha通过了考试。类似地，如果我们没有改变Aisha的成绩，那么我们将收到两次一致的查询结果，即可推导出Aisha没有通过考试。

综合情况3和4，我们可以得到结论，仅靠禁止对个体敏感属性的查询是无法完全解决敏感属性的隐私泄露问题的，情况3展现了一种需要数据特例来辅助的泄露情况，而情况4中使用的攻击方法即为**差分攻击**。

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。

### 差分隐私

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，使得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。我们仍然以一个例子来说明这种处理过程。

表5.2 医疗数据示例

|  |  |
| --- | --- |
| **姓名** | **是否患癌症** |
| Tom | 是 |
| Bob | 否 |
| Amy | 是 |
| Lily | 否 |
| Alice | 是 |

表5.2展示了一组医疗数据，其包括病人的姓名和是否罹患癌症的统计信息。当外部的研究人员想要针对这组数据展开分析研究时，医院需要为其提供统计查询服务，但是不能泄露具体个体是否罹患癌症的信息。我们假设一个基本的查询函数，该查询是针对给定的查询条件来查询数据集前条数据中罹患癌症的人数的。

显而易见，如果我们不对数据做任何处理，当攻击者试图知道Alice是否罹患癌症时，只需执行查询和，此时通过即可知晓Alice是否罹患癌症，形成了差分攻击。

为了抵御差分攻击，我们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，我们可以得到新的。

假设我们产生的，结合数据集，此时,，且由于我们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，攻击者将无法再通过对两次查询的差值来获取Alice是否罹患癌症的隐私信息，达到了保护个体隐私的目的，即**能够提供差分隐私**。我们称能提供了随机化输出的算法为一个**满足差分隐私的算法**。

接下来，我们对差分隐私进行形式化定义。

定义为域中的元素，从中抽取个的集合组成数据集，其中属性的个数为维度。表示数据集D的集合，即从中抽取个组成的数据集的集合。若两个数据集和具有相同的属性结构，二者之间只有一条记录不同，则称和为相邻数据集（Neighboring Datasets）。设算法为一个随机算法，*R*为实数域。

**定义5.1 差分隐私。**若随机算法，对于任意子集，在任意两个相邻数据集上，满足

则称随机算法满足差分隐私。

参数即表示随机算法不满足差分隐私的概率，。当时，称随机算法满足差分隐私。-差分隐私又称为纯差分隐私(Pure Differential Privacy)，而-差分隐私又称为近似差分隐私(Approximate Differential Privacy)。

在差分隐私中，我们知道，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[, ]之间，也就是说，隐私参数可以用来控制算法在两个相邻数据集上输出相同结果的概率比值，体现了差分隐私的保护水平。越小表示隐私保护水平越高，相反数值越大表示隐私损失越大。当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，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，的取值也反映了数据的可用性，在普通情况下，越小，数据可用性越低。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。

### 差分隐私性质

差分隐私保证：如果数据分析者除了数据分析任务之外不能对数据集进行额外查询，就无法增加数据集中每条记录的隐私损失。也就是说，如果我们使用随机性算法保护了个人的隐私，那么数据分析者就不能仅通过背景知识以及算法的输出来增加隐私损失，无论是在正式定义中，还是在任何直观意义上。形式化来说，与数据集无关的映射与一个满足差分隐私的算法组合起来，仍然满足差分隐私，我们将其称为后处理不变性 (Post-Processing) 。

**定理5.1 后处理不变性.** 若随机算法满足-差分隐私，对于任意随机映射，则是差分隐私的。

在现实的应用场景中，往往需要多次对数据集进行查询来满足日常的数据分析任务。在这种情况下，差分隐私的串行组合性(Sequential Composition)和并行组合性(Parallel Composition)提供了在多次使用随机算法查询数据集时计算隐私损失的方法。

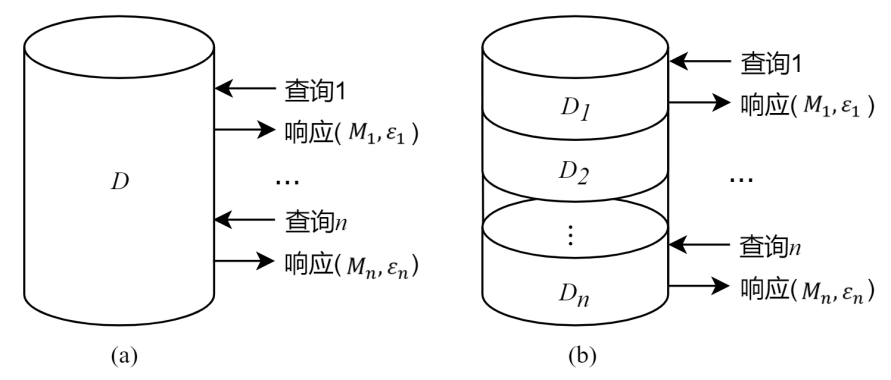


图5.2 差分隐私的组合性质：(a)串行组合性，(b)并行组合性

**定理5.2 串行组合性.** 给定一组差分隐私算法，每个满足-差分隐私。利用这组机制对相同的数据集进行查询，组合算法满足-差分隐私。

串行组合性表明用一组差分隐私算法查询同一个数据集后，数据集中每条记录的隐私损失将不超过全部所有算法导致的隐私损失的总和。然而，将随机算法的隐私损失简单求和的方式往往并不尽如人意，因为这样会导致最终的隐私损失中更为重要的参数偏大。

在实际应用中，学界一般更惯出差分隐私参数的大小，而的取值是一个较小的值即可。因此，一些学者研究放宽对参数的限制，减小的取值的合成定理。高级合成定理(Advanced Composition)考虑了隐私参数的期望（一阶矩）的上界，Moment Accountant方法则考虑隐私参数的矩生成函数（所有阶矩）的上界，这些方法都通过对的略微放大，大幅缩小了隐私参数的上界。串行组合性在现实场景中应用较广，一些论文中简称为组合定理（composition theorem）。

当一组算法处理彼此不相交的数据集，那么这一组算法序列构成的组合算法的差分隐私保护水平取决于其中保护水平最差者，即隐私损失最大的算法，该性质称为并行组合性，其形式化定义如下。

**定理5.3 并行组合性.** 给定一组差分隐私机制，每个提供差分隐私保证。若集合包含个不相交的子集，将这组算法分别作用在这些子集上，构成的组合算法会提供差分隐私。

相比于串行组合性，差分隐私的并行组合性更容易理解。由于差分隐私度量的是数据集中每一条记录的隐私损失，当随机算法不查询该数据集时，数据集中每条记录不损失隐私。在并行组合性中，每条记录只有在随机算法查询该数据集时损失隐私，差分隐私又是度量最差情况下隐私损失的定义，因此并行组合性即对应所有算法中隐私损失最差的算法。

### 差分隐私模型

基于差分隐私保护的数据发布是差分隐私研究中的核心内容，其目的是在不披露任何个人记录的情况下向公众输出汇总信息。

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。

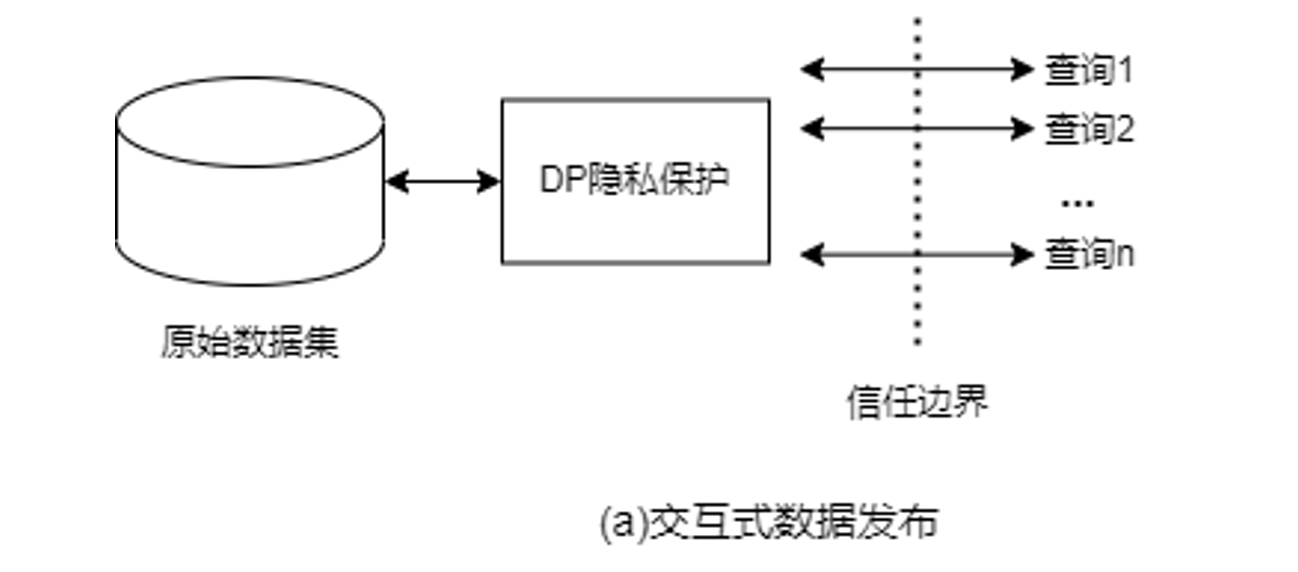
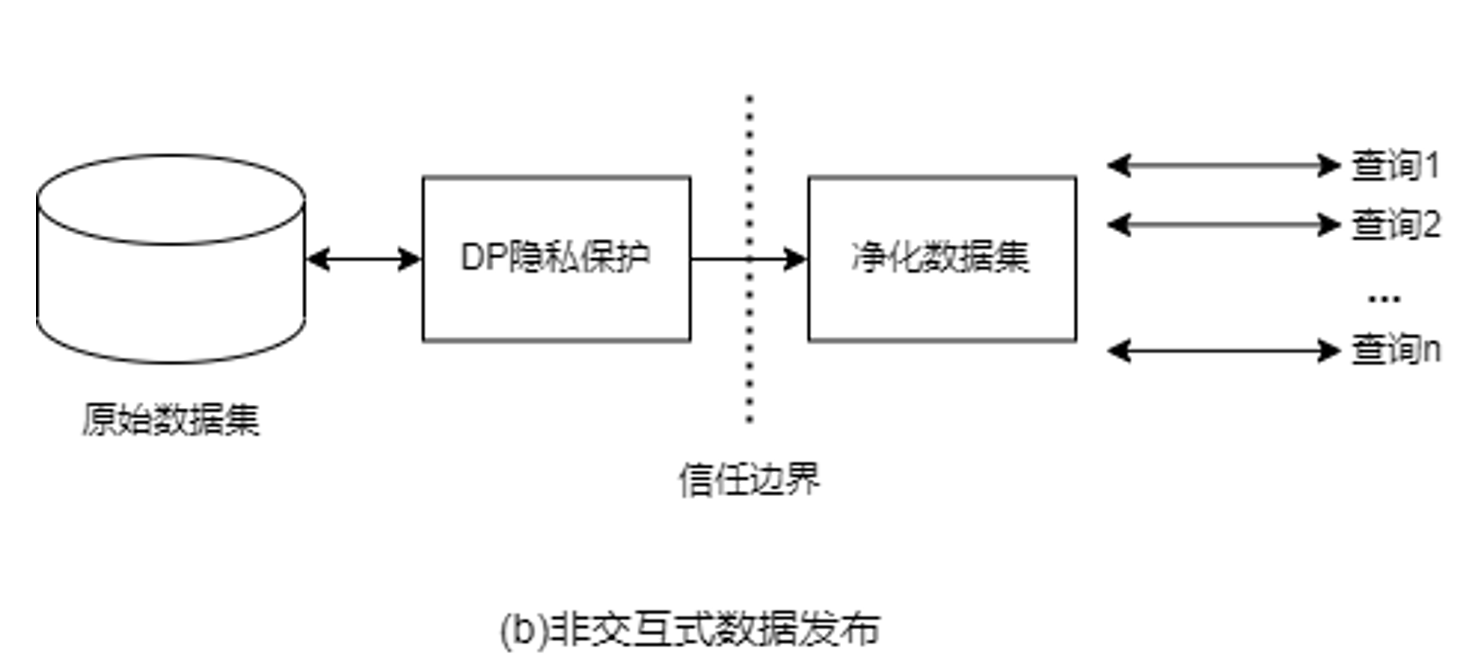
 

图5.5 数据发布模型

在交互式数据发布中，给定数据集和查询集，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𝐹中的查询，直到耗尽全部隐私预算。即在发布前一个查询的答案之前不能发出查询。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，给定数据集和查询集，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。

## 拉普拉斯机制

不同的差分隐私机制适用于不同类型的查询之中。一般来说，对数据库的查询可分为两个类型，即数值查询和非数值型查询。常使用的差分隐私机制主要有三种，即拉普拉斯(Laplace)机制、指数机制和随机响应机制。

拉普拉斯机制适合于数值查询；指数机制适合于非数值查询；随机响应机制适合于客户端信息收集中的噪音添加。

### 拉普拉斯机制

**1. 全局灵敏度**

考虑这样的SQL查询：

SELECT COUNT(\*) FROM D WHERE Sick = “糖尿病”

这是一个典型的计数查询，显然会受到差分攻击的影响。对于数值型的查询结果，拉普拉斯机制是在输出的数值上直接添加噪音。不过，在加入噪音之前，我们也要考虑另一个因素的影响。考虑在当前数据集只改动一条数据形成的相邻数据集，对于当前这个查询而言，如果我们只改动一条数据的情况下，那前述的SQL查询的结果改动至多为1。

但是，如果我们将查询变成这样：

SELECT 3\*COUNT(\*) FROM D WHERE Sick =“糖尿病”

改动一条数据的情况下，这个查询的结果可能会改变3而不是1了。为了解决类似的问题，**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**。

形式化地，有：

**定义5.2 全局灵敏度.** 设有查询函数，输入为一个数据集，输出为维实数向量。对于任意相邻数据集和，则

称为函数𝑓的全局灵敏度。其中是和之间的-阶范数距离，记为灵敏度。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（即距离之差的绝对值），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。

**2. 拉普拉斯分布**

拉普拉斯机制通过向查询结果或原始数据加入服从拉普拉斯分布的噪声来实现差分隐私。那么，为什么会选择拉普拉斯分布呢？

该分布的概率密度函数可参见图5.1，其中横轴表示随机变量x，纵轴表示相对应的概率密度（具体纵轴输出的是 出现的概率大小），b是尺度参数。我们可以看到，拉普拉斯分布的特点都是在函数中值当*x*=0时，其输出最大；而在两侧，其输出呈指数型下降，且是对称的，满足这种特点的分布是对称无偏的。这个特点使得随机化算法的噪声是无偏的（即数学期望是0），进而使得噪声的均值为0，即使得水平得以大大降低，能提高数据有效性。

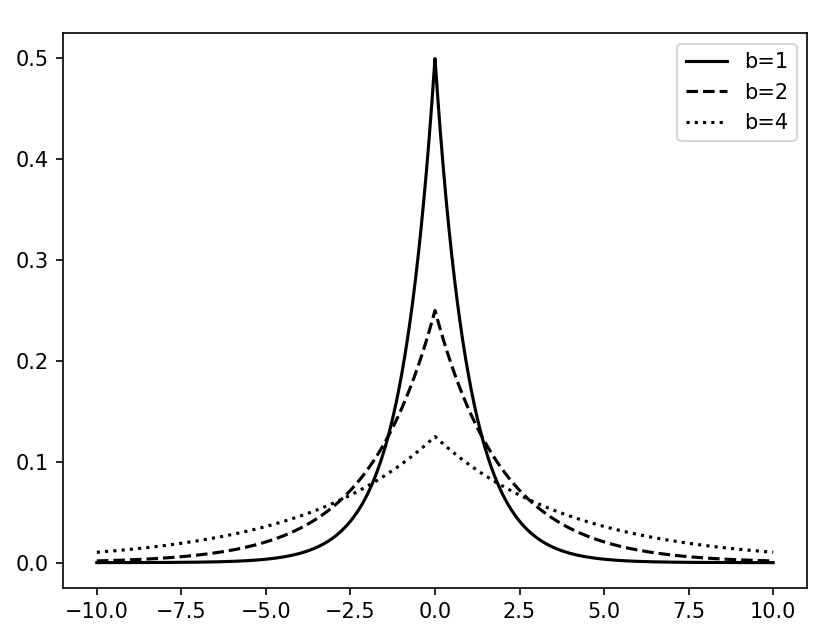


图5.1 Laplace概率密度函数

**3. 拉普拉斯机制**

我们使用来表示位置参数为0、尺度参数为𝑏的拉普拉斯分布，表示在尺度参数为*b*的时候输出结果为*x*的概率，即概率密度函数，表示为：

根据差分隐私的理论，在灵敏度为1的时候，加入的噪声参数满足为，即能满足-差分隐私。下面，我们来形式化地定义拉普拉斯机制。

**定理5.4 拉普拉斯机制** 设函数，为灵敏度。给定数据集，则随机算法

能提供-差分隐私保护。其中，是一维的输出域，为独立的、服从尺度参数*b*为的Laplace分布的随机变量。

如图5.1所示的不同参数的拉普拉斯分布还可以看出，在灵敏度不变时，越小，引入的噪声越大。

接下来，本章给出一个能够生成拉普拉斯噪音，并基于该噪音设计差分隐私方案。本部分代码由C语言实现，并在Ubuntu系统中进行了测试。

### 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：

若随机变量服从于离散分布,即,同时有服从于，取，则有

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, 使得

2. 在时，产生具有分布函数的随机变量

该算法首先以概率来选择一个子分布函数然后取的随机数来作为的随机数。

而具体到拉普拉斯机制而言，其概率密度分布函数为：

其均值为0，方差为。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和,即

2. 计算

其中，为拉普拉斯机制的隐私参数。在本例中，由于敏感度为1，其值为。

我们将该部分的代码放置于文件“laplace.c”中。函数uniform\_data给出了一个生成服从均匀分布的随机数的算法。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：

double t;

    \*seed = 2045.0 \* (\*seed) + 1;

    \*seed = \*seed - (\*seed / 1048576) \* 1048576;

    t = (\*seed) / 1048576.0;

线性同余法的公式为.此处我们设定。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，我们将映射到区间上，就完成了一个服从的随机数生成。

t = a + (b - a) \* t;

    return t;

有了这个算法，我们就可以利用此前推导的公式来生成服从拉普拉斯分布的随机数了。函数laplace\_data以隐私预算和随机数种子作为输入，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函数产生两个服从的随机数：

double u1,u2, x;

    u1 = uniform\_data(0.0, 1.0, seed);

    u2 = uniform\_data(0.0, 1.0, seed);

之后，我们将这两个随机数代入到此前的公式中：

if (u1 < 0.5)

    {

        x = beta \* (log(2\*u1)+u2);

    }

    else

    {

        x = u2 - (beta \* log(2\*(1-u1)));

    }

这样，我们就可以得到一个以隐私参数和随机种子生成的服从拉普拉斯分布的随机数了。

### 统计查询应用

**实验5.1 对某类数值型数据统计查询的基于拉普拉斯机制的防护方案。**

实验内容：对一个数据集（zoo.csv）进行统计查询，该数据集描述了一个动物园喂食的场景，第一列中数据为动物名称，第二列中数据为动物每天消耗的胡萝卜数量。查询定义为“每日进食超过55根胡萝卜的动物数量”。请设计相关的隐私保护方案，确保查询过程不泄露信息。

**1. 方案一（交互式发布）：对查询返回的结果添加噪音**

考虑一种现实：重复推测攻击。

重复攻击是针对差分隐私的攻击方式。因为拉普拉斯机制添加噪声的特点是无偏估计，多次查询后均值为0，如果攻击者向数据库请求重复执行同一个查询语句 ，将结果求和平均，就有极大的概率获得真实结果。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 | 10000 |
| 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 |

事实上，隐私预算刻画了一类查询任务的总体允许的隐私泄露程度。如果仅仅将作为生成拉普拉斯噪音的参数的话，如上表所示，多次查询很容易就实现隐私信息的推断。

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，允许的查询次数为*k*，则每次查询分配的预算为，这样才能达到-差分隐私的目标。

因此，对于统计查询而言，如果在查询结果上进行反馈，则需要定义所能支持的次数，进而按上述方式对每次查询进行预算的分配。换句话说，这种添加噪音的方式，会使得每次查询都会消耗一定的隐私预算，直到隐私预算都被消耗干净，就再也不能起到保护的作用。

**2. 方案二（非交互式）：对数据集中的数据添加噪音**

不在结果上加噪音，而是数据上加噪音，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，我们对每条记录都根据设定的隐私预算产生相关的拉普拉斯噪声，并添加，进而生成合成的数据集。

本部分实现置于testraw.c中。首先设计了一种能够读取csv文件并对其进行分析的算法csv\_analysis，来实现对csv文件的读取和其上属性的处理。该算法在第19行完成了对csv数据集的读取后，会进入一个循环体，对数据集内的数据条目进行循环处理和加噪并判断该条数据加噪后是否符合前述查询，并输出该查询的结果：

while(original\_data[i].name)

    {

        x = laplace\_data(beta,&seed); //产生拉普拉斯随机数x作为噪音

        printf("Added noise:%f\t%s %d\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

}

在主函数中，指定全局敏感度为1，以10和0.1作为两个隐私预算，并生成基于时间的随机种子：

long int seed;

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

    double eps[]={10, 0.1}; //指定两个隐私预算，分别代表极大，极小

    srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand方法）

int i = 0;

为了刻画信息泄露的情况，我们可以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。我们利用这两个隐私预算分别基于原始数据集和相邻数据集生成加噪数据并进行前述查询，以此来对噪声的影响进行展示和比较。

    while(i<2)

    {

        printf("Under privacy budget %f, sanitized original data with animal name and laplace noise:\n",eps[i]);

        double beta = sen / eps[i]; //拉普拉斯机制下，实际公式的算子beta为敏感度/预算

        seed = rand()%10000+10000; //随机种子产生

        csv\_analysis("./zoo.csv",beta,seed); //先调用原始数据集

        printf("==================Using neighbour dataset==================\n");

        seed = rand()%10000+10000; //随机种子更新

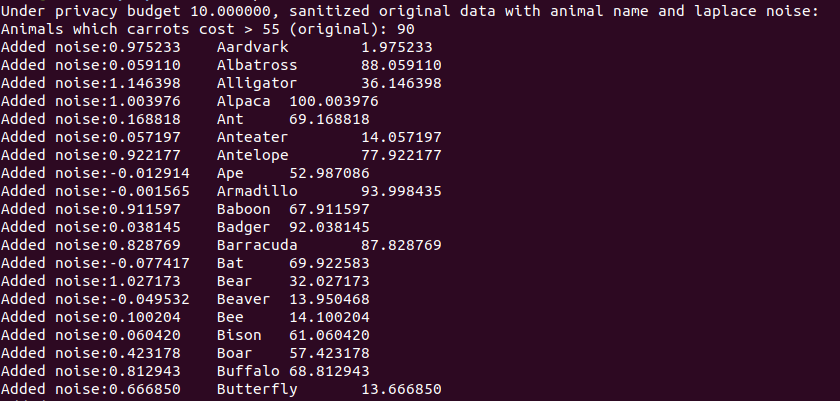
        csv\_analysis("./zoo\_nb.csv",beta,seed); //再调用相邻数据集

        printf("===========================================================\n";

        i++;

    }

该算法的运行结果如下。我们先输入了较大的隐私预算（10），此时，观察生成的噪音和加噪后的数据与原始数据的差别：



可以看到，在投入较大的隐私预算的情形下，添加的噪音均小于1或略大于1。对于特定查询“每日进食大于55根胡萝卜的动物个数”，在该预算下，加噪前和加噪后的响应一致，数据可用性好：

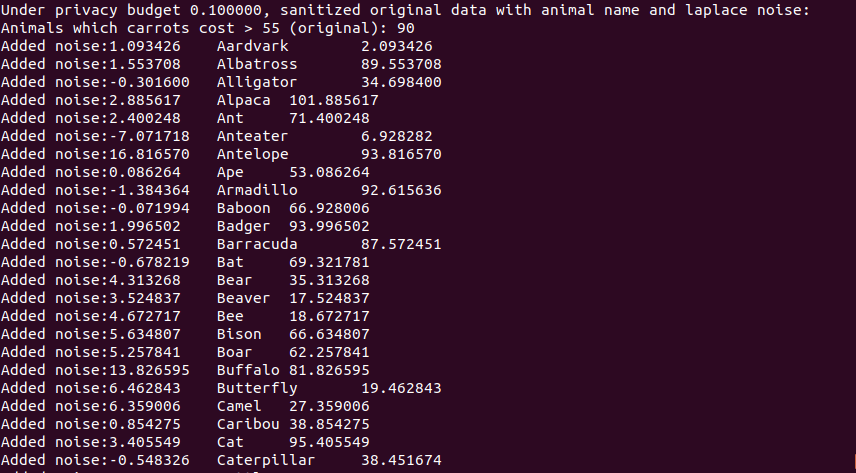


但是，观察标记后对相邻数据集的处理情况，我们可以发现，加噪后数据集对该查询的响应仍与数据集的变化一致，均为89，体现出了Dugong离开数据集造成的差异，不能有效抵御对该查询的差分攻击。





我们此时再输入0.1作为隐私预算。可以看到，在该预算下，产生的拉普拉斯噪音也增大了，这使得加噪后的查询结果也受到了影响：





但是同时，观察对相邻数据集进行加噪的结果，可以发现，虽然相邻数据集的直接查询结果受到了“Dugong”项移除的影响，但加噪后的相邻数据集查询结果与加噪前相比变化巨大，不再能反映出“Dugong”项移除的影响：



即，投入较少的隐私预算时，虽然数据的可用性降低了，但是能够更好地抵御差分攻击的影响。这里需要说明的是，由于算法存在一定随机性且比较简单，所以在某些时候可能无法取得上述结果，可以多试几次。

### 直方图发布应用

**实验5.2 对直方图发布的基于拉普拉斯机制的防护方案。**

直方图发布也是差分隐私数据发布的一类重要应用。一个简单的医疗数据场景：假设其建立于某医疗数据集medicaldata.csv之上，其第一列数据为一特定年龄区间，第二列数据为该年龄区间患有某种疾病的人数。要发布的直方图即是以第一列数据为直方图的桶（Bucket）发布的统计数据。

在本例中，数据集medicaldata.csv和其相邻数据集md\_nb.csv，描述了md\_nb.csv是在medicaldata.csv的基础上，将其中“30-40”区间的统计值-1而产生的，模拟原始医疗数据集内一名患者退出数据共享。

本部分的代码实现位于testhist.c。首先，与数值型应用实现类似，我们定义了一个csv\_analysis函数来实现对csv文件的读取和其上属性的处理。该算法在文件第20行完成了对csv数据集的读取后，会进入一个循环体，对数据集内的数据条目进行循环处理和加噪，并输出该区间数据加噪后的结果：

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 Histobuckets \* original\_data = NULL;

    original\_data = hb\_csv\_parser(original\_file);

    int sum=0,i=0;

    double x = 0;

    while(original\_data[i].bucket)  //循环为原始数据集内各桶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].bucket,original\_data[i].count+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

        i++;

    }

}

在主函数中，我们仍然指定全局敏感度为10和0.1两个隐私预算，并生成基于时间的随机种子：

long int seed;

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

    double x;

    srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand方法）

    double eps[]={10,0.1};

此后，我们利用这两个隐私预算分别基于原始数据集和相邻数据集来进行加噪的直方图发布，以此来对噪声的影响进行展示和比较。

while(i<2)

    {

        printf("Under privacy budget %f, sanitized original bucket with laplace noise:\n",eps[i]);

        double beta = sen / eps[i]; //拉普拉斯机制下，实际公式的算子beta为敏感度/预算

        seed = rand()%10000+10000; //随机种子产生

        csv\_analysis("./medicaldata.csv",beta,seed); //先调用原始数据集

        printf("==================Using neighbour dataset==================\n");

        seed = rand()%10000+10000; //随机种子更新

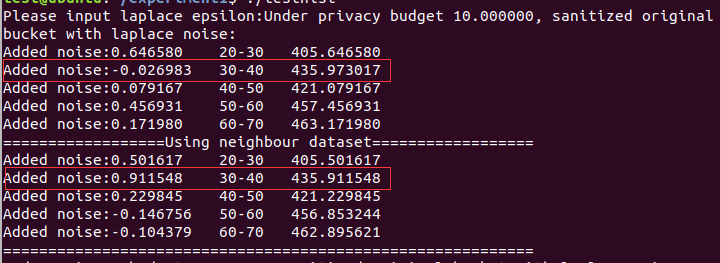
        csv\_analysis("./md\_nb.csv",beta,seed); //再调用相邻数据集

        printf("===========================================================\n");

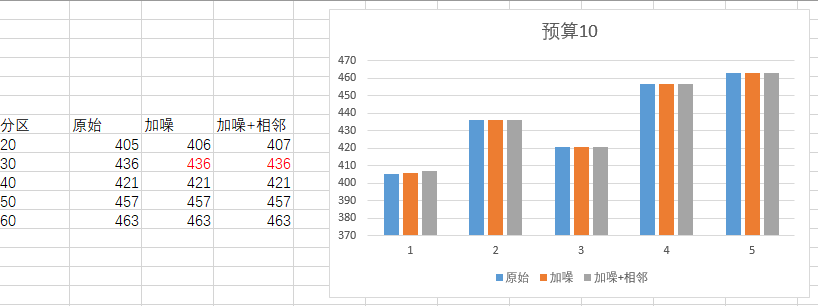
        i++;

    }

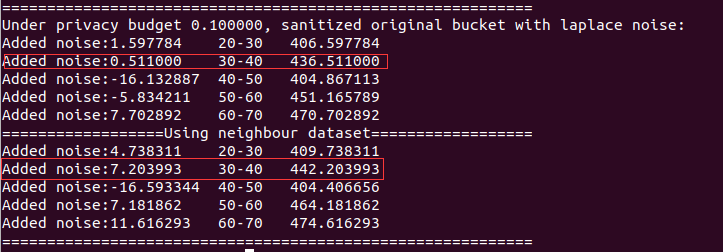
该算法的运行结果如下。我们先输入了较大的隐私预算（10），此时，观察生成的噪音和加噪后的数据与原始数据的差别：



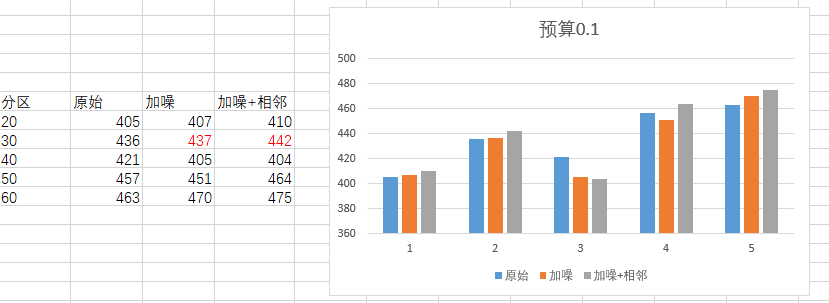
可以看到，当隐私预算为10时，由于加入噪音量级较小，相邻数据集的变化仍能被体现。我们将这些数据输入到excel中，得到使用预算为10的差分隐私算法生成的数据和直方图如下：



此时，使用0.1为预算的实验结果为：



可以看到，由于噪音规模的提高，在相邻数据集的变化影响下，查询结果不减反增。即，虽然数据可用性变差，但能保护实际数据的变化不被攻击者获取，可抵御差分攻击。使用预算0.1的差分隐私算法生成的数据和直方图如下：



## 指数机制

### 适用场景

拉普拉斯机制适用于数值型查询结果，但是对于一些特别的查询，直接对查询返回的结果直接加入随机噪音可能会造成较大的影响。

**对于一些特别的查询**，直接对查询返回的结果直接加入随机噪音可能会造成较大的影响。

**举例来说：**将南瓜卖给Alice、Bob、Charlie，其中Alice和Bob最多能接受每个南瓜1元，而Charlie则可以接受每个南瓜3元。此时，我们想要定下价格，让我们受益最大。

如果我们定价为1元，这个价格在每个人的预算内，那我们能够获得3元（1+1+1）；如果我们定价3元，那只有Charlie能买得起，那我们也可以获得3元的收益。此时我们可以发现，如果我们试图为最优定价加价，我们有两个最优定价可选，即1元和3元。

**为了保护我们的定价，可以加入噪音，但是当加入了一个噪音时，即使这个噪音极小，也会大幅度影响我们的收益。**当我们定价为1.01元时，只有Charlie仍然愿意购买，我们只能收入1.01元。而当我们定价3.01元时，Charlie也不会购买，我们的收益会变成0元。

这类查询(从1和3两个定价中选一个）和非数值类查询的特点类似，其查询结果是离散的、不能变动的数据。对于这类查询（查询结果为实体对象，如一种选择或一个方案），我们通常使用指数机制来进行处理。

### 工作原理

指数机制可以理解为对于某一问题的隐私抽样算法：对于一个问题, 存在一个打分函数，为数据集的输出域中的每一个可能的输出进行打分，指数机制会以打分函数为基础为输出分配概率来对结果进行抽样。

指数机制的目标就是降低数据集中任意记录的改变对打分函数的影响，从而限制函数的输出导致数据集中记录的隐私泄露。下面，我们来形式化定义指数机制。

**定理5.5 指数机制** 设随机算法输入为数据集，抽样输出为一实体对象，为打分函数，代表打分函数𝑞(𝐷,𝑟)的灵敏度。当算法满足

提供差分隐私。

可以理解为，是从输出域中抽样输出一个实体对象*r*的随机算法，抽样概率正比于。

我们再给出一个指数机制的应用实例。假如某学校要举办一场体育比赛，可供选择的项目来自集合{足球，排球，篮球，网球}，参与者们为此进行了投票，现要从中确定一个项目，并保证整个决策过程满足差分隐私。

对每一个参与者，他的投票选择都是个人隐私，不能泄露他喜欢哪种球类。诚如我们所知，差分攻击者可以通过两次输出的结果，来判断其中一个用户的喜好。指数机制将依据计算得到的抽样概率进行抽样输出，抽样概率使得差分攻击者难以有绝对优势猜测出攻击者到底投了哪一个项目。以得票数量为打分函数，显然。依据指数机制计算得到抽样概率如表5.2所示。

当预算为0时，四个项目的概率被均分了，指数机制无论抽样得到哪个输出，差分攻击者都不能获得任何有用的信息。而在较大时，打分最高的选项被输出的概率被放大，反之当较小时，各个选项在打分上的差异被平抑，其输出概率随着的减小而趋于相等。

表5.2 指数机制应用实例

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 | 打分函数 | 概率 | | |
|  |  |  |
| 足球 | 30 | 0.25 | 0.424 | 0.924 |
| 排球 | 25 | 0.25 | 0.330 | 0.075 |
| 篮球 | 8 | 0.25 | 0.141 | 1.5e-05 |
| 网球 | 2 | 0.25 | 0.105 | 7.7e-07 |

## 随机响应机制

### 适用场景

前两种机制都属于中心化差分隐私的范畴，即数据收集者是可信的，用户发送给数据收集者的数据一定是真实的，随机性的添加由数据收集者完成。而随机响应机制认为数据收集者并不可信，其随机性的添加在用户发送数据前就完成了，即数据收集者收集到的数据已经是被用户扰动过的。在现实生活中，各手机厂商或是各软件厂商的大数据分析就是类似的场景，我们并不能完全信任收集我们数据的厂商，所以我们需要随机响应机制来进行保护。目前，谷歌、苹果、小米等大型厂商已经开始了对随机响应机制的使用。

### 随机响应机制

随机响应机制的主要思想是利用对敏感问题响应的不确定性对原始数据进行隐私保护，保证不同的单个记录之间的不可区分性，其随机性存在于用户向数据收集者提交数据之前。

随机响应技术主要包括两个步骤：扰动性统计和校正。

关于扰动性统计，我们以一个例子来介绍。以询问用户是否患某种病症为例，假设有个用户，其中患病的真实比例为，但数据收集者并不知道。收集者希望对比例进行统计，于是发起一个敏感问题的询问，即“你是否患病”，每个用户对该问题进行响应，用1表示“是”，0表示“否”，第个用户给出答案，但出于隐私性考虑，用户不直接响应真实结果，而是通过某种方式来随机决定是否输出这个结果。

这种随机性有多种实现方式。如本节此前所述，一种最简单的实现方式是用户可以丢一枚硬币，并根据一次或者多次投掷，并界定这些投掷结果的排列组合对应的输出来实现随机化，如两次投掷时，“正正”输出1，“正反”输出0等。

还有一种类似的例子是采用掷骰子的方式来实现随机化。对于是否输出自己真实的疾病情况的调查，我们要求用户掷一个骰子，若结果为1则再掷一次，结果为4-6则返回0。第二次投掷时，若其结果为1则返回1，其他情况返回0。显而易见，这种二次掷骰方法输出1的概率为1/6\*1/6=1/36。

通过迭代掷骰和设计输出规则，我们可以实现满足任意概率的输出方式，满足任意定义。

从更高层次来看，无论是使用多次掷硬币，还是多次掷骰，其均等同于借助于一枚非均匀的硬币来给出答案，而其正面向上的概率为，反面向上的概率为。抛出该硬币，若正面向上则回答真实答案，反面向上则回答相反的答案。

在扰动性统计过程中，利用上述方法对个用户的回答进行统计，可以得到患者人数的统计值。假设统计结果中，回答“是”的人数为，则回答“否”的人数为。显然按照上述统计，回答“是”和“否”的用户比例分别为

然而，我们会发现，上述统计比例并非对真实比例的无偏估计，因此还需要对统计结果进行校正。此时我们构建以下似然函数：

得到的极大似然估计：

以下关于 的数学期望保证其为真实分布的无偏估计

由此可以得到校正后的统计值，患病人数估计值为。

综上，根据总人数、回答患病的人数和扰动概率，即可得到真实患病人数的统计值。为保证其满足本地差分隐私，其中。

接下来，我们形式化地给出随机响应机制的定理：

**定理5.7 （）随机响应机制** 设随机算法输入为记录 ，输出为 ，随机算法会输出一个伯努利随机变量，其中：

则算法满足本地差分隐私。

上述为基本的二值随机响应技术，对于输入域更大的情况有如直接编码(Direct Encoding)、直方图编码(Histogram Encoding)、一元编码(Unary Encoding)等方法。而在真实使用场景中，为了达到更好的隐私保护效果不直接使用这些技术，如谷歌提出的RAPPOR方法，结合布隆过滤器进行两次随机响应；还有苹果的Private Count Mean Sketch(CMS)方法将结合了Sketch技术与随机响应。由于本地差分隐私在实际场景中更容易被用户所接受，因此对于本地差分隐私技术的研究和应用近年来的热度保持在较高的水平。