第 6 章 差分隐私

学习要求：了解差分攻击，掌握差分隐私的概念、性质，了解差分隐私模型；掌握拉普拉斯机制，了解拉普拉斯机制在统计查询和直方图发布中的应用。理解指数机制的机理和适用场景。掌握随机响应机制；了解差分隐私的新型应用，包括 ESA 模型、数据合成等。

课时：2 课时/4 课时

建议授课进度：2 课时 [6.1~6.2]、[6.5]

4 课时 [6.1]、[6.2~6.3]、[6.4]、[6.5]

差分隐私概念最早由Cynthia Dwork等人[1]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值ε。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。

6.1 基本概念

6.1.1 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6-1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（如，所有女生）的成绩查询请求**。

表 6-1 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。

如果说查询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，查询3中的隐私泄露不复存在，因为此时查询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。

[1] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, “Calibrating noise to sensitivity in private data analysis,” in Theory of Cryptography: Third Theory of Cryptography Conference, 2006, pp. 265-284.

（4）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，第一个查询的结果是4，第二个查询的结果是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。

综合查询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，查询3展现了一种需要数据特例来辅助的泄露情况，而查询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。

6.1.2 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，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6-2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$f(i) = count(i)$，该查询针对给定的查询条件i来查询数据集D前i条数据中罹患癌症的人数。

表 6-2 医疗数据示例

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

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$f(5)$和$f(4)$，此时通过$f(5) - f(4)$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$f'(i) = count(i) + noise$。

假设我们产生的$noise \in [-1,1]$，结合数据集，此时$f'(5) = [2,4], f'(4) = [1,3]$，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，攻击者将有概率无法再通过两次查询的差值来获取Alice是否罹患癌症的隐私信息，达到一定程度上保护隐私的目的，即**能够提供差分隐私**。我们使用差分隐私来量化一个随机化算法提供多强的隐私保护。当随机化算法满足差分隐私的定义时，我们称该算法为**差分隐私算法**。

接下来，我们对差分隐私进行形式化定义。

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。

定义6-1（差分隐私） 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

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

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ε −差分隐私。ε −差分隐私又称为纯差分隐私（Pure Differential Privacy），而(ε, δ) −差分隐私又称为近似差分隐私（Approximate Differential Privacy）。

在ε −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$\varepsilon$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。

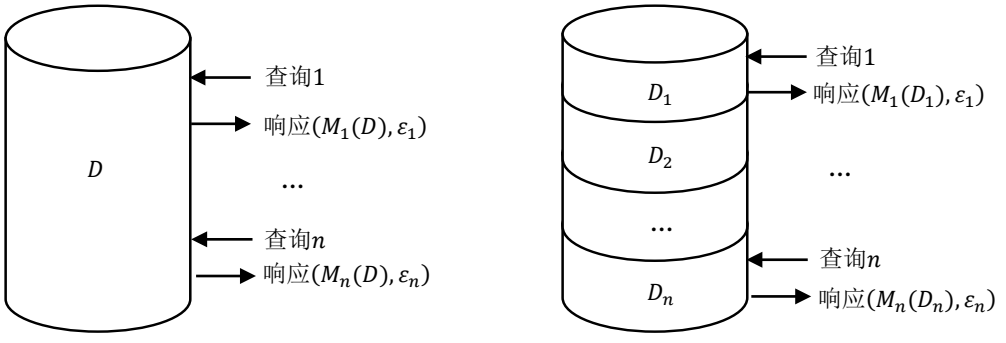
从另一方面来看，ε的取值也反映了数据的可用性，在普通情况下，ε越小，数据可用性越低。ε越大，隐私保护越弱，但数据可用性越高。在一些场景中，必须在ε取值较大的情况下，才能实现有效的数据分析或模型训练任务。因此，隐私预算ε的取值需要结合实际场景和需求，在输出结果的隐私性和可用性之间进行权衡。

6.1.3 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f与一个满足(ε, δ) −差分隐私的算法M组合起来，仍然满足(ε, δ) −差分隐私，我们将其称为后处理不变性（Post-Processing）。

定理 6-1（后处理不变性） 若随机算法$M: X^n \to Z$满足(ε, δ) −差分隐私，对于任意随机映射$f: Z \to Z'$，则$f \circ M: X^n \to Z'$是(ε, δ) −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，如图 6-1 所示。



（a）串行组合性　　（b）并行组合性

图 6-1 差分隐私的组合性质

定理 6-2（串行组合性） 给定一组差分隐私算法$M_1, M_2, \dots, M_n$，每个M_i满足$(\varepsilon_i, \delta_i)-$差分隐私。利用这组机制对相同的数据集$D$进行查询，组合算法$M(D)=(M_1(D), M_2(D), \dots, M_n(D))$满足$(\sum \varepsilon_i, \sum \delta_i)-$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，因为这样会导致隐私参数ε偏大。

在实际应用中，学界一般更关注约束差分隐私参数ε的大小，而δ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δ的限制，减小ε的取值的合成定理。高级合成定理[2]（Advanced Composition）考虑了隐私参数ε的期望（一阶矩）的上界，Moment Accountant 方法[3]则考虑隐私参数ε的矩生成函数（所有阶矩）的上界，这些方法都通过对δ的略微放大，大幅缩小了隐私参数ε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。

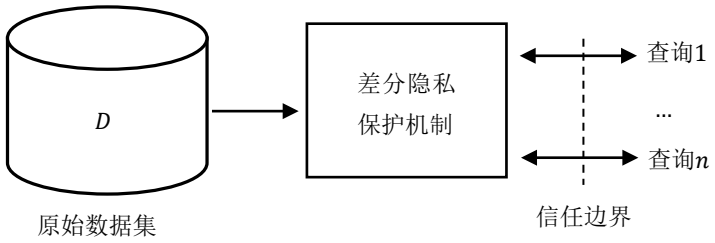
定理 6-3（并行组合性） 给定一组差分隐私机制$M_1, M_2, \dots, M_n$，每个M_i提供$(\varepsilon_i, \delta_i)-$差分隐私保证。若集合$D$包含$n$个不相交的子集$D = D_1 \cup D_2 \dots \cup D_n$，将这组算法分别作用在这些子集上，构成的组合算法$M(D)=(M_1(D_1), M_2(D_2), \dots, M_n(D_n))$会提供$(max\ \varepsilon_i, max\ \delta_i)-$差分隐私。

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。

6.1.4 差分隐私模型

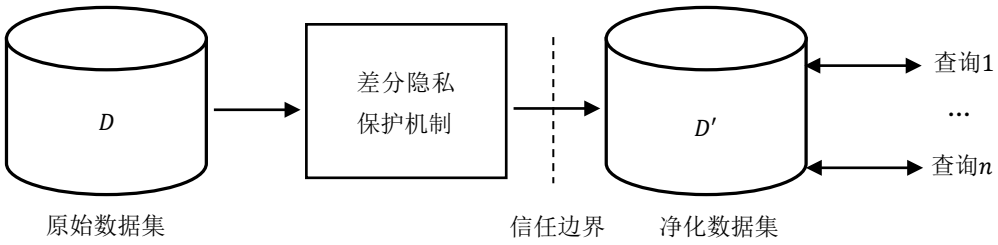
基于差分隐私保护的数据发布是差分隐私研究中的核心内容，其目的是在不披露任何个人记录的情况下向公众输出汇总信息。

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，如图 6-2 所示。



（a）交互式数据发布方案

[2] D. Cynthia, G. N. Rothblum, and S. Vadhan, “Boosting and differential privacy,” in IEEE 51st Annual Symposium on Foundations of Computer Science, 2010, pp. 51–60.
[3] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in SIGSAC conference on computer and communications security, 2016, pp. 308-318.



（b）非交互式数据发布方案

图 6-2 数据发布模型

在交互式数据发布中，给定数据集D和查询集$F = \{f_1, \ldots, f_m\}$，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F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$F = \{f_1, \ldots, f_m\}$，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。

6.2 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查询。针对这两类查询，我们分别介绍两种常用的差分隐私机制，适用于数值查询的拉普拉斯机制和适用于非数值查询的指数机制。除此之外，我们在本章中介绍了一种适合于客户端信息收集的本地化隐私保护机制，即随机响应机制。

6.2.1 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D只改动一条数据形成的相邻数据集D'，对于当前这个查询而言，如果我们只改动一条数据，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**。

定义 6-2（全局灵敏度） 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。

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？

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。

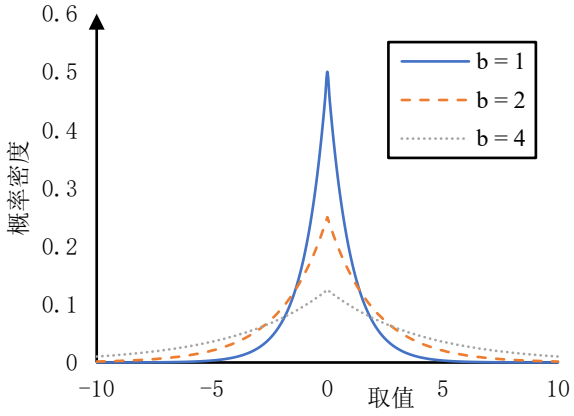


图 6-3 拉普拉斯分布概率密度函数

3. 拉普拉斯机制

我们使用$Lap(b)$来表示位置参数为 0、尺度参数为b的拉普拉斯分布，$Lap(x|b)$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：

$$Lap(x|b) = \frac{1}{2b}\exp(-\frac{|x|}{b})$$

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b为$\frac{1}{\varepsilon}$，即能满足ε −差分隐私。下面，我们来形式化地定义拉普拉斯机制。

定理 6-4（拉普拉斯机制） 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。

从图 6-3 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，ε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，并基于该噪音设计的差分隐私方案。本部分代码由 C 语言实现，并在 Ubuntu 系统中进行了测试。

6.2.2 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$P(\zeta = \mathrm{i}) = p_i$,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$P(\zeta = i) = p_i, (i = 1,2,3 \dots, K)$；

（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。在本例中，由于数据集的敏感度为 1，其值为$1/\epsilon$ 。

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$U(0,1)$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：

```
1. double t;
2. *seed = 2045.0 * (*seed) + 1;
3. *seed = *seed - (*seed / 1048576) * 1048576;
4. t = (*seed) / 1048576.0;
```

线性同余法的公式为$X_{n+1} = aX_n + c \ mod \ m$. 此处我们设定$a = 2045, m = 1048576, c = 1$。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。

```
1. t = a + (b - a) * t;
2. 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：

```
1. double u1,u2, x;
2. u1 = uniform_data(0.0, 1.0, seed);
```

```
3. 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：

```
1. if (u1 < 0.5)
2. {
3.     x = beta * (log(2*u1)+u2);
4. }
5. else
6. {
7.     x = u2 - (beta * log(2*(1-u1)));
8. }
```

这样，我们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。

6.2.3 统计查询应用

实验 6-1 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

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。

表 6-3 交互式发布的概率表

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	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ε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ε作为生成拉普拉斯噪音的参数的话，如表 6-3 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ε，允许的查询次数为k，则每次查询分配的预算为$\frac{\varepsilon}{k}$，这样才能达到ε −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，按照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，会使每次查询都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

该方案不在结果上加噪音，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，我们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。

本部分实现置于 testraw.c 中。为了对 csv 文件执行操作，首先需要设计一种能够读取 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：

```
1. while(original_data[i].name)
2. {
3.     x = laplace_data(beta,&seed); //产生拉普拉斯随机数 x 作为噪音
4.     printf("Added
   noise:%f\t%s %d\t%f\n",x,original_data[i].name,original_data[i].carro
   ts+x); //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); //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：

```
1. long int seed;
2. int sen = 1;  //对于一个单属性的数据集，其敏感度为 1
3. double eps[]={10, 0.1}; //指定两个隐私预算，分别代表极大，极小
4. srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand 方法）
5. 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。

```
1. while(i<2)
2. {
3.     printf("Under privacy budget %f, sanitized original data with
   animal name and laplace noise:\n",eps[i]);
4.     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
5.     seed = rand()%10000+10000; //随机种子产生
6.     csv_analysis("./zoo.csv",beta,seed); //先调用原始数据集
7.     printf("====Using neighbour dataset====\n");
8.     seed = rand()%10000+10000; //随机种子更新
9.     csv_analysis("./zoo_nb.csv",beta,seed); //再调用相邻数据集
10.    printf("================================\n";
11.    i++;
```

```
12. }
```

我们先输入了较大的隐私预算（10），此时，生成的噪音和加噪后的数据与原始数据的差别如图 6-4（a）所示。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，如图 6-4（b）所示。

```
Under privacy budget 10.000000, sanitized original data with animal name and laplace noise:
Animals which carrots cost > 55 (original): 90
Added noise:0.975233    Aardvark        1.975233
Added noise:0.059110    Albatross       88.059110
Added noise:1.146398    Alligator       36.146398
Added noise:1.003976    Alpaca  100.003976
Added noise:0.168818    Ant     69.168818
Added noise:0.057197    Anteater        14.057197
Added noise:0.922177    Antelope        77.922177
Added noise:-0.012914   Ape     52.987086
Added noise:-0.001565   Armadillo       93.998435
Added noise:0.911597    Baboon  67.911597
Added noise:0.038145    Badger  92.038145
Added noise:0.828769    Barracuda       87.828769
Added noise:-0.077417   Bat     69.922583
Added noise:1.027173    Bear    32.027173
Added noise:-0.049532   Beaver  13.950468
Added noise:0.100204    Bee     14.100204
Added noise:0.060420    Bison   61.060420
Added noise:0.423178    Boar    57.423178
Added noise:0.812943    Buffalo 68.812943
Added noise:0.666850    Butterfly       13.666850
```

（a）不使用 DP 的情况和使用 DP 的输出

```
Animals which carrots cost > 55 (Under DP): 90
```

（b）使用 DP 的查询结果

图 6-4 预算为 10 时的输出数据集和查询结果

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。此时的查询结果如图 6-5 所示。

```
==================Using neighbour dataset==================
Animals which carrots cost > 55 (original): 89
```

（a）不使用 DP 的情况

```
Animals which carrots cost > 55 (Under DP): 89
============================================================
```

（b）使用 DP 的情况

图 6-5 预算为 10 时的对相邻数据集的查询结果

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，如图 6-6 所示。

```
Under privacy budget 0.100000, sanitized original data with animal name and laplace noise:
Animals which carrots cost > 55 (original): 90
Added noise:1.093426    Aardvark        2.093426
Added noise:1.553708    Albatross       89.553708
Added noise:-0.301600   Alligator       34.698400
Added noise:2.885617    Alpaca  101.885617
Added noise:2.400248    Ant     71.400248
Added noise:-7.071718   Anteater        6.928282
Added noise:16.816570   Antelope        93.816570
Added noise:0.086264    Ape     53.086264
Added noise:-1.384364   Armadillo       92.615636
Added noise:-0.071994   Baboon  66.928006
Added noise:1.996502    Badger  93.996502
Added noise:0.572451    Barracuda       87.572451
Added noise:-0.678219   Bat     69.321781
Added noise:4.313268    Bear    35.313268
Added noise:3.524837    Beaver  17.524837
Added noise:4.672717    Bee     18.672717
Added noise:5.634807    Bison   66.634807
Added noise:5.257841    Boar    62.257841
Added noise:13.826595   Buffalo 81.826595
Added noise:6.462843    Butterfly       19.462843
Added noise:6.359006    Camel   27.359006
Added noise:0.854275    Caribou 38.854275
Added noise:3.405549    Cat     95.405549
Added noise:-0.548326   Caterpillar     38.451674
```

（a）不使用 DP 的情况和使用 DP 的输出

```
Animals which carrots cost > 55 (Under DP): 99
```

（b）使用 DP 的查询结果

图 6-6 预算为 0.1 时的输出数据集和查询结果

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，如图 6-7 所示。

```
Animals which carrots cost > 55 (Under DP): 95
============================================================
```

图 6-7 预算为 0.1 时的相邻数据集查询结果

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较为简单，所以在某些时候可能无法取得上述结果，可以多试几次。

6.2.4 直方图发布应用

实验 6-2 对直方图发布的基于拉普拉斯机制的防护方案。

在探索直方图发布的差分隐私应用场景中，以一个具体的医疗数据集为例，其数据框架建立在一个文件（例如 medicaldata.csv）之上，其中首列记录了不同的年龄段，而第二列则统计了在这些年龄段中，患有特定疾病的人数。我们打算发布的直方图，基于第一列数据作为区间分布的基础，展示了各个年龄段对应的疾病患病率。

在这一过程中，直方图查询的工作机制是将整个数据集细分成若干个互不重叠的部分，每部分对应一个年龄区间，并计算每个区间内的数据点数量。鉴于这些区间之间是完全独立的，即数据集中的任何单一记录的变动仅会影响到一个特定的区间，因此，这类查询的敏感度被定为 1。基于这一特点，在每一区间的查询结果上添加根据隐私预算采样的噪声，就能够确保满足ε −差分隐私的标准。通过这种方法，既保护了数据的隐私安全，又能够在一定程度上准确地反映出不同年龄段的疾病分布情况，这对于公共健康研究和政策制定具

有重要的参考价值。

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，模拟了一个场景：即一个患者决定退出医疗数据共享计划。

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：

```
1. void csv_analysis(char* path, double beta, long int seed)
2. {
3.     FILE *original_file = fopen(path,"r+"); //读取指定路径的数据集
4.     struct Histobuckets * original_data = NULL;
5.     original_data = hb_csv_parser(original_file);
6.     int sum=0,i=0;
7.     double x = 0;
8.     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
9.     {
10.         x = laplace_data(beta,&seed); //产生拉普拉斯随机数
11.         printf("Added noise:%f\t%s\t%f\n",x,original_data[i].bucket,original_data[i].count+x); //当投入较少预算时，可能会出现负数
12.         i++;
13.     }
14. }
```

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：

```
1. long int seed;
2. int sen = 1;  //对于一个单属性的数据集，其敏感度为 1
3. double x;
4. srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand 方法）
5. 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。

```
1. while(i<2)
2. {
3.     printf("Under privacy budget %f, sanitized original bucket with laplace noise:\n",eps[i]);
4.     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
5.     seed = rand()%10000+10000; //随机种子产生
6.     csv_analysis("./medicaldata.csv",beta,seed); //先调用原始数据集
7.     printf("=======Using neighbour dataset========\n");
8.     seed = rand()%10000+10000; //随机种子更新
```

```
9.      csv_analysis("./md_nb.csv",beta,seed); //再调用相邻数据集
10.     printf("=====================================\n");
11.     i++;
12. }
```

该算法的运行结果如图 6-8 所示。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：

```
Please input laplace epsilon:Under privacy budget 10.000000, sanitized original
bucket with laplace noise:
Added noise:0.646580    20-30   405.646580
Added noise:-0.026983   30-40   435.973017
Added noise:0.079167    40-50   421.079167
Added noise:0.456931    50-60   457.456931
Added noise:0.171980    60-70   463.171980
==================Using neighbour dataset==================
Added noise:0.501617    20-30   405.501617
Added noise:0.911548    30-40   435.911548
Added noise:0.229845    40-50   421.229845
Added noise:-0.146756   50-60   456.853244
Added noise:-0.104379   60-70   462.895621
============================================================
```

图 6-8 预算为 10 时的输出

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图 6-9 所示：

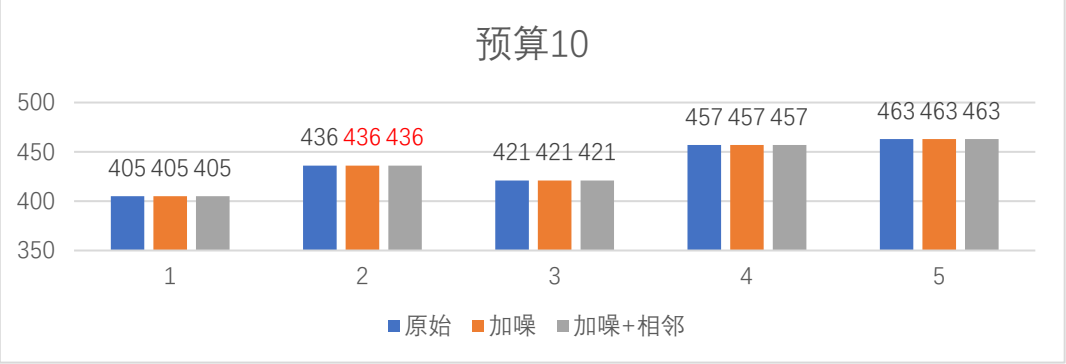


图 6-9 预算为 10 时的直方图

使用 0.1 为预算的实验结果如图 6-10 所示。

```
============================================================
Under privacy budget 0.100000, sanitized original bucket with laplace noise:
Added noise:1.597784    20-30   406.597784
Added noise:0.511000    30-40   436.511000
Added noise:-16.132887  40-50   404.867113
Added noise:-5.834211   50-60   451.165789
Added noise:7.702892    60-70   470.702892
==================Using neighbour dataset==================
Added noise:4.738311    20-30   409.738311
Added noise:7.203993    30-40   442.203993
Added noise:-16.593344  40-50   404.406656
Added noise:7.181862    50-60   464.181862
Added noise:11.616293   60-70   474.616293
============================================================
```

图 6-10 预算为 0.1 时的输出

随着噪声量的增加，我们可以观察到一个现象：尽管数据的可用性受到影响，结果却意外地显示出增长，而非减少。这表明，即便数据的准确性下降，我们依然能够有效隐藏

数据集之间的微小变化，从而防止攻击者利用这些变化进行差分攻击。使用预算 0.1 的差分隐私算法生成的数据和直方图如图 6-11 所示：

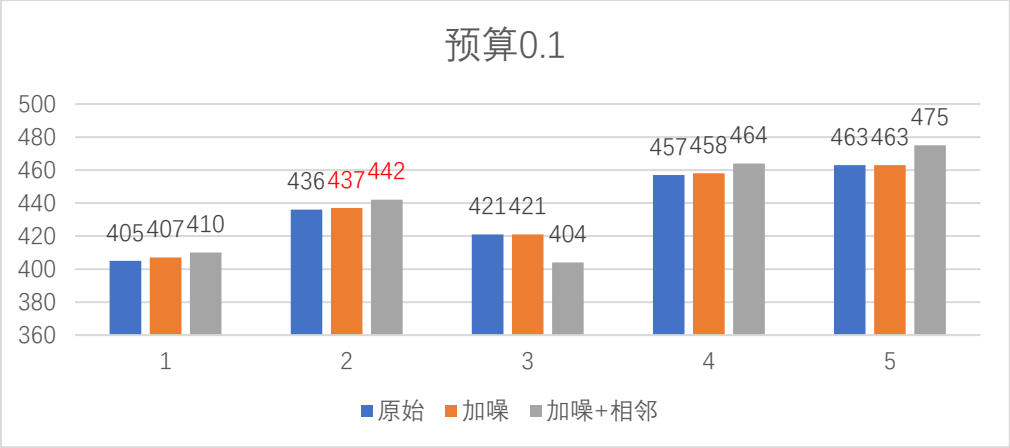


图 6-11 预算为 0.1 时的直方图

6.3 指数机制

6.3.1 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**，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，让我们收益最大。

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 [4]来进行处理。

6.3.2 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q, 存在一个打分函数$q(D,r) \to R$，为数据集D的输出域O中的每一个可能的输出r进行打分，指数机制会以打分函数为基础为输出 r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D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D中记录的隐私泄露。下面，我们来形式化定义指数机制。

定理 6-5（指数机制） 设随机算法M输入为数据集D，抽样输出为一实体对象$r \in O$，$q(D,r)$为打分函数，Δq代表打分函数$q(D,r)$的灵敏度。当算法M满足

$$M(D,q) = \left\{ r \mid \Pr[r \in O] \propto \exp\left(\frac{\varepsilon \times q(D,r)}{2\Delta q}\right) \right\}$$

[4] F. McSherry, and K. Talwar. “Mechanism design via differential privacy,” in 48th Annual IEEE Symposium on Foundations of Computer Science, 2007, pp. 94-103.

提供ε −差分隐私。

指数机制可以理解为，$M(D,q)$是从输出域O中抽样输出一个实体对象r的随机算法，抽样概率正比于$\exp\left(\frac{\varepsilon\times q(D,r)}{2\Delta q}\right)$。

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ε −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$\Delta q = 1$。依据指数机制计算得到抽样概率如表 6-4 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ε较大时，打分最高的选项被输出的概率被放大，反之当ε较小时，各个选项在打分上的差异被平抑，其输出概率随着ε的减小而趋于相等。

表 6-4 指数机制应用实例

项目	打分函数 $\Delta q = 1$	概率		
		$\varepsilon = 0$	$\varepsilon = 0.1$	$\varepsilon = 1$
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

6.4 随机响应机制*

6.4.1 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。此类在用户本体提供随机性扰动的场景被称为本地化差分隐私，本地化差分隐私可以视为差分隐私中数据集只有一条数据的特例。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。

6.4.2 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。

1. 扰动性统计

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n个用户，其中患病的真实比例为π，但数据收集者并不知道。收集者希望对比例$\hat{\pi}$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i个用户给出答案X_i，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，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ε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p，反面向上的概率为$1-p$。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n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n_1，则回答“否”的人数为$n-n_1$。显然按照上述统计，用户回答“是”或“否”的概率分别为

$$\Pr[X_i=1]=\pi p+(1-\pi)(1-p)$$
$$\Pr[X_i=0]=(1-\pi)p+\pi(1-p)$$

2. 校正

显然，由于掷硬币的不均匀性影响，我们此时获得的统计比例与我们期望得到的真实统计比例π与$1-\pi$并不相等，并非是无偏统计。因此，我们需要对统计结果进行校正。构建以下似然函数：

$$L=\big(\pi p+(1-\pi)(1-p)\big)^{n_1}\big((1-\pi)p+\pi(1-p)\big)^{n-n_1}$$

得到π的极大似然估计：

$$\hat{\pi}=\frac{p-1}{2p-1}+\frac{n_1}{(2p-1)n}$$

以下关于$\hat{\pi}$的数学期望保证其为真实分布π的无偏估计

$$E(\hat{\pi})=\frac{1}{2p-1}\left(p-1+\frac{1}{n}\sum_{i=1}^{n}X_i\right)=\frac{1}{2p-1}\big(p-1+\pi p+(1-\pi)(1-p)\big)=\pi$$

由此可以得到校正后的统计值，患病人数估计值为$N=\hat{\pi}\times n=\frac{p-1}{2p-1}n+\frac{n_1}{2p-1}$。

综上，根据总人数n、回答患病的人数n_1和扰动概率，即可得到真实患病人数的统计值。

3. 实例

以性别的频数统计为例，我们来验证一下上面的推导。

原始数据：

——男：16，女：4

概率：

——0.8 概率上报真实性别，0.2 的概率上报假的性别

统计结果：

——男：14

——女：6

校正：

——男：$[20*(0.8-1)+14]/(0.8*2-1)\approx 16.7$

——女：$[20*(0.8-1)+6]/(0.8*2-1) \approx 3.3$

从统计学上，校正后的频数统计结果和真实值是相近的。

4. 随机响应机制

接下来，我们形式化地给出随机响应机制的定理：

定理 6-6（（二值）随机响应机制） 设随机算法M输入为记录$x \in \{0,1\}$，输出为$y \in \{0,1\}$，随机算法M会输出一个伯努利随机变量$M(x)$，其中：

$$Pr[M(x)=x]=\frac{e^{\varepsilon}}{e^{\varepsilon}+1}$$

$$Pr[M(x)=1-x]=\frac{1}{e^{\varepsilon}+1}$$

则该算法满足ε－本地化差分隐私。

因为$\frac{Pr[M(x)=x]}{Pr[M(x)=1-x]}=e^{\varepsilon}$，并且$Pr[M(x)=x]+Pr[M(x)=1-x]$ =1，为满足ε－本地化差分隐私，$\varepsilon=\ln\frac{p}{1-p}$。因此，在确定隐私预算$\varepsilon$之后，就可以确定$p$的取值，进而可以明确随机响应执行的过程。

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。由于本地化差分隐私在实际场景中更容易被用户所接受，因此对于本地化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。

6.5 差分隐私应用*

6.5.1 Google RAPPOR 模型

随机聚合隐私保护顺序响应算法 [5]错误!未找到引用源。（Randomized Aggregatable Privacy-Preserving Ordinal Response, RAPPOR）是谷歌于 2014 年提出的本地化差分隐私算法。谷歌考虑数据收集者会重复收集用户数据的场景，如果不断通过随机响应上传随机值的话，用户的隐私将会不断的损失。为了缓解此问题，谷歌提出了两阶段随机响应算法，具体的本地随机化方案如图 6-12 所示：

[5] Ú. Erlingsson, V. Pihur, and Al. Korolova, “Rappor: Randomized aggregatable privacy-preserving ordinal response,” in 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1054-1067.

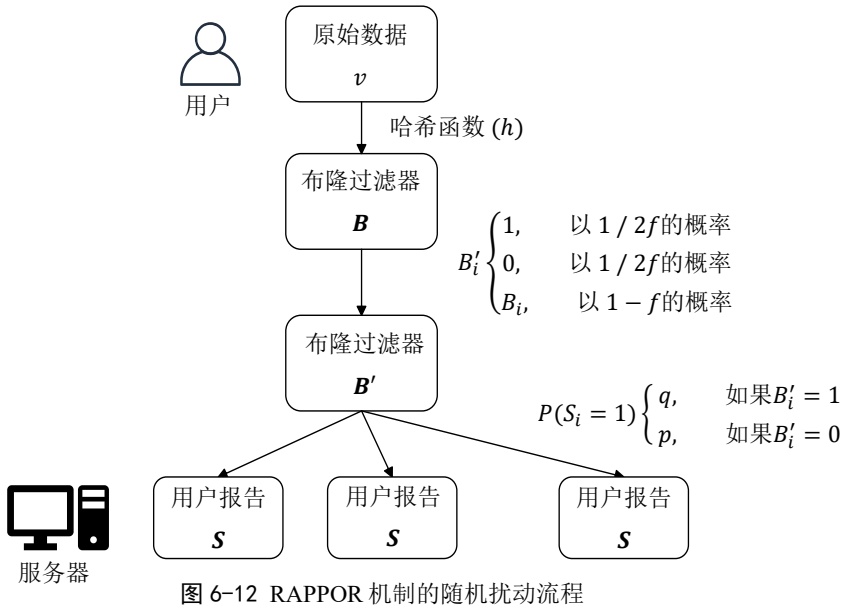


图 6-12 RAPPOR 机制的随机扰动流程

（1）用户将它们的真实值v通过h个哈希函数，映射到k维布隆过滤器之中，作为真实值的对应向量$\boldsymbol{B}$。

（2）在得到向量$\boldsymbol{B}$之后，用户会对向量$\boldsymbol{B}$中的每一位进行一次永久的随机响应（每一位以$1-1/2f$的概率保持不变），得到伪向量$\boldsymbol{B'}$。随后，用户会永久保存伪向量$\boldsymbol{B'}$，作为真实值v的映射来使用。

（3）在服务器收集用户本地数据时，用户将会对伪向量$\boldsymbol{B'}$中的每一位进行随机响应（如果$B'_i=1$，则随机响应后为 1 的概率为q; 如果$B'_i=0$，则随机响应后为 1 的概率为p），并将随机响应后的向量$\boldsymbol{S}$发送至服务器。

与一般的随机响应机制相比，RAPPOR 机制的隐私性更强。考虑服务器多次收集用户的数据时，随机响应算法会不断的通过真实值来生成随机值，每次上传随机结果都会消耗隐私。而对于 RAPPOR 机制来讲，上传的向量S是由伪向量B'来决定的。从差分隐私的后处理不变性来理解，服务器得到的输出，都是来自于伪向量B'的，即使它收集到了足够的信息，也只能将B'恢复出来，而无法恢复真实值v对应的向量B。因此，在 RAPPOR 中，用户得到的隐私保护更为稳定，不会面临隐私耗尽的风险。

6.5.2 Google ESA 模型

分布式的数据收集场景，在实际中应用尤为广泛。如苹果及谷歌公司所使用的差分隐私技术都是分布式场景下，使用本地化差分隐私技术，在用户本地进行数据扰动而后进行数据收集和分析。本地化差分隐私中不需要可信的数据收集者，用户上传的数据已经满足差分隐私保护，这种信任假设和隐私保护方式能够更好的被用户所接受，可以提高用户参与数据统计分析活动。然而，对于数据分析者来说，数据的可用性也尤为重要。通过本地化差分隐私技术，每一个用户都向数据中添加了噪声，而大量用户的数据扰动行为会导致数据最终的误差变大。以计数查询为例，与中心化差分隐私算法相比，为了达到相同的隐私保护水平，本地化差分隐私的误差为中心化算法的$O(\sqrt{n})$倍。在谷歌对其本地算法的后续研究中发现，本地化差分隐私机制需要海量的数据才能提供较为准确的信息，并且噪音往往是非常大的——10 亿用户数据中，百万量级的信号都有可能会被淹没。

为了在分布式数据收集场景，数据收集者不可信的情况下，提高数据的可用性，谷歌在本地化差分隐私机制的基础上提出了 ESA[6]（Encode-Shuffle-Analyze）模型，我们简称为

[6] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnés, and

洗牌模型（shuffle model）。相较于本地模型，洗牌模型在用户和不可信的数据收集者之间引述了一个洗牌器（shuffler），如图 6-13 所示。在洗牌模型中，洗牌器将用户扰动后的数据进行混洗，以打乱用户与数据之间的对应关系。对于数据收集者来说，洗牌器的工作提供了额外的随机性，使得本地模型中用户的输出与用户本身的联系不复存在。学者发现，这种随机性可以有效提高差分隐私机制的保护效果，即洗牌器提供了隐私放大（Privacy Amplification）作用。简单来讲，即用户在客户端使用了满足ε −本地化差分隐私机制，并将输出的密文发送给洗牌器，洗牌器经过随机洗牌后，算法将提供$\left(\varepsilon', \delta'\right)$ −差分隐私，且$\varepsilon' < \varepsilon$。这种隐私放大作用带来的最直接的好处就是能够提高数据的可用性，即达到几乎相同的隐私保护水平时，机制添加了更少的噪音。

洗牌模型不仅能减少差分隐私机制的噪音，还具有容易拓展的优点，原有的本地差分隐私机制，均能通过将输入发送至洗牌器而获得隐私放大的效果，从而降低因隐私保护而添加的噪音。这使得洗牌模型可以简单的部署到现有的数据收集场景中。

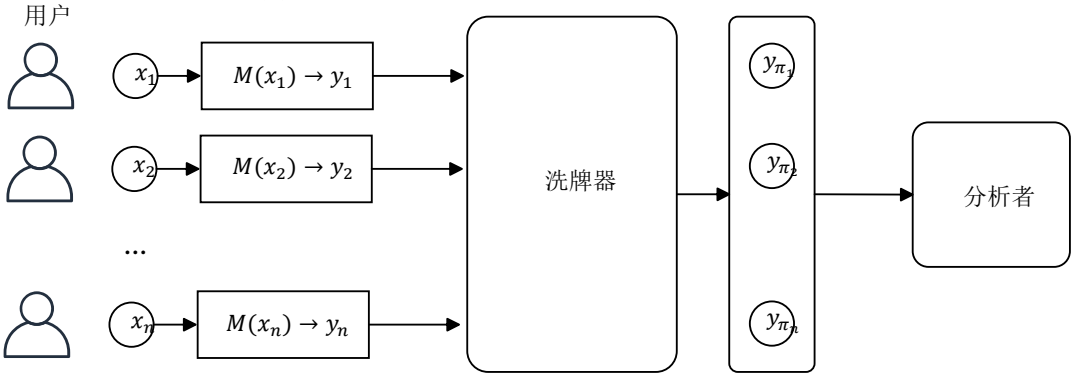


图 6-13 洗牌模型

6.5.3 基于图模型的数据合成

数据合成是一种非交互式差分隐私模型，它旨在合成与原始数据相似但不完全相同的合成数据集，使得其在保护原始数据的隐私的同时，能够支持理论无限次的查询而不会产生额外的隐私泄露累积问题，且对同一个查询能产生完全一致的查询。该方法常用于需要对数据执行大量查询，且对查询结果的一致性有较高要求的场景，如敏感领域大数据分析（如医疗、金融、人口信息等）、数据分析外包多方共用数据分析等场景下的保护隐私的数据分析服务。

1. 原始数据集和合成数据集

合成数据生成算法的输入是一个原始数据集，其输出是与原始数据集“形状”一致（即列数和行数相同）的合成数据集。在此基础上，我们希望合成数据集的数据与原始数据集的对应数据满足相同的性质，即响应相同查询时能够输出相似的结果。例如，如果我们选择某个人口普查数据集作为原始数据集，我们希望合成产生的数据集与原始数据集在年龄分布上具有相似的人群表现，且列之间的相关性能够得到保留（如年龄和从事职业的相关性）。

为了保证合成的数据集具有与原始数据集类似的“统计特征”（即对同样的查询能返回相近的结果），一般来说，要生成合成数据集，我们需要先确定原始数据集上各属性值的概率分布情况，而分布情况的确定主要是通过对原始数据集执行属性查询的方式来进行的，如“查询两种性别的占比情况”。此外，由于直接学习原始数据集的真实分布仍然会遭受差

Bernhard Seefeld, “Prochlo: Strong privacy for analytics in the crowd,” in the 26th Symposium on Operating Systems Principles, 2017, pp. 441-459.

分攻击的影响，为了保证差分隐私，一种较为常用的技术是在合成数据算法对原始数据集进行查询时，利用部分隐私预算对查询结果进行扰动。

下面我们使用一个单列的数据集来演示合成数据集的产生过程。我们以一个拥有单属性“年龄”的“初中学生信息”数据表为例，如表 6-5 所示。

表 6-5 初中学生数据表

记录	年龄
1	12
2	13
3	15
4	13
5	15
6	12

要生成一个合成数据集，我们首先可以对 12-15 岁间的年龄的计数值定义为一个直方图查询，并计算每个年龄段的人数。此后，我们向得到的计数查询结果添加拉普拉斯噪音，并对加噪后的结果进行归一化，使得所有计数的和为 1。进而，这些归一化的结果可以被视为每个属性值在原始数据表中出现的概率。此时，我们可以使用这些概率进行采样，根据各属性值对应的概率值来产生随机的属性值。一个可能的合成数据表如表 6-6 所示。

表 6-6 合成的数据表

记录	年龄
1	13
2	15
3	12
4	15
5	12
6	13

本例中所展示的合成数据算法较为简单，且仅对一列数据执行了数据合成。而实际的合成数据算法中会对所有列的统计特征进行扰动，噪音的添加方法也会更加精密，会根据选中查询的属性域的大小及用户定义的特定查询的重要性（即权重）等指标调整加入到不同属性分布的隐私预算。有些算法还会在多个不同的步骤中使用隐私预算添加噪音。

2. 数据集的图模型化表示

接下来，我们将对生成合成数据集的具体算法进行介绍。当前的数据合成算法中较为常用的方法主要有两类，一类是基于机器学习的算法，一类是基于图模型的算法。其中，基于机器学习的算法虽然能有效分析和学习数据集中各属性的分布，但是往往需要大量性能开销和复杂的超参数选择过程。因此，目前基于图模型的合成算法是数据合成算法的主流。

根据具体算法不同，图模型算法会使用不同的图模型来表示和推理数据集中的属性关联性，我们这里以一种基于贝叶斯网络的图模型方案来举例，如图 6-14 所示。该方案以年龄作为初始属性，构建了一张有向无环图（Directed Acyclic Graph, DAG）。在该图中，每个图节点均是一种属性，并使用有向边对 A 中属性的条件独立性进行建模。

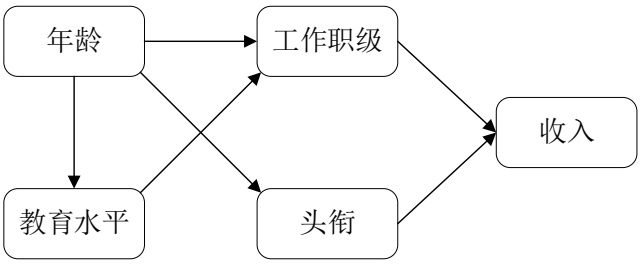


图 6-14 基于贝叶斯网络的数据集图化表示

显然，对于数据合成而言，想要获得原始数据集的概率分布时，直接查询并计算的出整个数据集的联合分布最为准确。但在实际算法执行过程中属性个数较多时，其属性域的大小将呈指数级增长，计算这类高维度分布的开销极高，不具备实用性。而将数据集上的各属性间的关联构建成一个图模型可使得我们通过度量图模型中部分低维属性的联合分布后，即可利用信念传播等数学工具来推理出未被度量的属性分布，为合成数据集提供参考。

3. 从图模型到合成数据

那么，如何选择合适的低维联合分布查询来构建图模型估计原始数据集的分布，并最终生成合成数据呢？当前，基于图模型的合成数据算法主要分为两类，一类是批处理式算法，一类是迭代式算法。这里我们针对迭代式算法来进行详细讲解。该算法主要分为两个主要阶段，即分布迭代优化阶段和基于分布的合成数据生成阶段。

1）分布迭代优化阶段

该阶段以原始数据集D和工作负载W作为输入。其中，工作负载W是由全体候选查询组成的集合，而候选查询通常由用户指定或根据特定规则生成。如当我们想要使用数据集上各属性间的二维分布来度量数据集的整体分布时，我们可以枚举出所有可能的属性间二维组合，生成对这些属性组合的联合分布的查询，由这些查询组成该场景下的候选查询集合。从图模型的角度来看，这也就是将该数据集上所有属性作为图模型的顶点时，所有该模型中可能出现的二维边缘。该阶段算法会在这些二维边缘上抽样部分边缘，利用原始数据集来计算这些边缘的分布。此后，使用这些边缘来推理出其他未抽样的边缘，最终估计出一个完整的属性分布并进行迭代优化。

具体来说，该阶段首先产生一个初始化的属性分布$\widehat{D_0}$，此后利用“选择-测量-估计”三个步骤来迭代优化这个分布。算法的迭代阶段使用了指数机制和高斯机制来为生成的分布合入噪音，最终产生对D的一个含噪估计分布$\widehat{D}$来用于数据合成。

2）数据合成阶段

该阶段利用前一阶段所估计出的含噪分布$\widehat{D}$内各属性值的概率来随机地生成一个合成数据集。由于在估计分布的过程中已经使用了基于差分隐私的算法，数据生成阶段无需再进行加噪处理。

接下来我们对“选择-测量-估计”三个步骤进行具体介绍。在每轮迭代的选择阶段，算法使用原始数据集和前一轮迭代所估计出的属性分布来对W中的查询进行响应，根据响应结果，使用部分隐私预算，利用指数机制来选择一个查询。该指数机制使用的打分函数常根据相同查询的两个响应的“差距”来打分，并倾向于输出本轮分布估计响应最差的查询，作为本轮的选择结果。选择每轮最差查询的原因是，由于执行的轮次有限，算法期望每轮都能尽可能多地对分布进行优化，因此使用了一种贪心的方式来对查询进行选择。

在测量阶段，算法对数据集再次执行选择阶段选中的查询，利用高斯机制或拉普拉斯机制在其响应结果上加噪并加入到观测序列。经过选择和测量阶段引入的差分隐私机制，算法在估计阶段产生的分布也是满足差分隐私的。

最后，估计阶段利用观测序列内所有被测量过的查询响应来更新分布中的对应边，使

用镜像梯度下降算法找寻一个能最为匹配已观测的结果的分布，并通过信念传播算法来推理图模型中各边缘的概率分布。每轮估计产生的分布将作为下轮选择阶段的选择依据，以迭代式地寻找当前估计分布响应较差的查询。具体迭代轮数常与数据集的属性个数相关，依具体算法而定。

基于图模型的数据合成算法能够在观测较少的属性间边缘分布的基础上对数据集进行图模型化并估计其总体分布，实现了合成数据精度和开销的权衡，使之能广泛利用于各类需要使用数据合成算法的场景之中。

课后习题

1. 有关差分隐私，下列说法正确的是（ ）

A. 交互式差分隐私方案理论上可以支持无限次查询，隐私预算不会随着查询而耗尽。

B. 数据合成一般对应非交互式的差分隐私模型。

C. 在为具体数据集设计差分隐私算法时，无需考虑一条数据的改动对查询结果会造成多大的改变。

2. 虽然差分隐私技术对于保护用户隐私具有良好效果，但现有大规模商用的差分隐私方案并不能很好地保护用户隐私。请尝试调研谷歌、苹果的差分隐私实现，来谈谈为什么会这样，以及如何解决这些问题？