剪烛随记:Differential Privacy

差分隐私简明学习笔记

作者：谢天

时间：Dec 29, 2022, 大二寒假

版本：1.0

学校：中科大网安学院

临渊羡鱼，不如退而结网

目录

序言

致亲爱的读者：

好吧，或许此份笔记的读者寥寥无几，但是我还是想写下来，因为我觉得这是一件很有意思的事情。

首先多亏了 Dwork 和 Roth 的 *Algorithmic Foundations of Differential Privacy*，其次还要感谢 Waterloo 大学 Gautumn 教授的 CS860 与 Vermort 大学的 CS211，本书借鉴上者良多。

在一切开始之前，请先谅解一下作者的水平，我还是一名大二本科生，抱着 Dwork 差分隐私洋文书独自猛啃。文中英文术语是完全凭借自己理解翻译的，其他图表之类则是参照了相关论文和资料。

在做笔记的过程中，越发意识到自己的不足，向上空间还是很大的。这份笔记是第一版，日后如果对差分隐私有更深刻的了解，我也会继续更新第二版，第三版…

你问我写这本书的意义？理由呢？总得有个坚持下去的动力吧。呃，我觉得其实万事不必需要那么多理由。就像有人得癌症，有人不想吃河鱼，有人喜欢把脑袋枕在胳膊上，就当上帝像一个小男孩，喜欢没事就在草地上踩蘑菇。你要问为什么？就是因为刚刚下过雨，草地里钻出无数的蘑菇，而他脚上正好有双不错的运动鞋而已。

阅读愉快！

谢天

第一章 The Spark of Differential Privacy

内容提要

- 历史背景
- Dinur & Nissim
- reconstruction attack
- Blatant Non-Privacy(BNP)

1.1 简述

本节从回溯 Dinur 的一篇论文开始，该文被认为是启发了 Dwork 差分隐私的灵感源泉。我暂时不会阐释怎样使得一个算法有多隐私，恰恰相反，我想说的是大部分方案都是“全然非隐私”的（BNP）。同时，我们会重点分析一种称为重构攻击（Reconstruction Attacks）的攻击形式，并阐述了为了保护隐私，我们应该增加的噪声理论上的值应该是多少。

注 本节均参照此论文 Revealing Information while Preserving Privacy Policy

1.2 简化版模型与基本表述

我们从这里严谨完备一个数据库模型，以下做了部分简化以便描述

- 定义每条数据为一行
- 每列为一个属性/特征
- 我们假定 Name,Postal Code,Date of Birth,Sex 等属性是不敏感的，是公开数据
- 只认为一种属性是敏感的，即 Has Disease？我们将其简化为 1bit(取值只能为 0 或 1)
- 令 $d \in \{0,1\}^n$ 表示隐私 bit 的向量

Name	Postal Code	Date of Birth	Sex	Has Disease?
Alice	K8V7R6	5/2/1984	F	1
Bob	V5K5J9	2/8/2001	M	0
Charlie	V1C7J	10/10/1954	M	1
David	R4K5T1	4/4/1944	M	0
Eve	G7N8Y3	1/1/1980	F	1

图 1.1: 数据库模型实例

命题 1.1 (模型概述)

这里假定有攻守双方，一方是查询者，一方是管理者。

查询者被允许提的问题形如：“数据库里有多少行在条件 X 下满足’Has Disease=1’?”

条件 X 可以是”Name= Alice OR Name = Charlie OR Name = David”，那么问题答案就是 2

更抽象起见，我们令查询向量为 $S = \{0,1\}^n$,0 表示该行不满足条件，1 表示满足条件，例如条件 X 可以表示为 $S = \{1,0,1,1,0\}$. 真实结果用 A(S) 表示，$A(S) = d \cdot S$ (例中 $A(S) = \{1,0,1,0,1\} \cdot \{1,0,1,1,0\} = 2$)

当然，这极大侵犯了隐私，所以管理者的响应 r(S) 会在 A(S) 的基础上加上一些噪声，界限小于等于 E

$$|r(S) - A(S)| \leq E \tag{1.1}$$

1.3 重构攻击的数学表达

定义 1.1 (全然非隐私 BNP)

我们称如下算法是全然非隐私的（blatantly non-private）：
如果攻击者能够重构一个数据库 $c \in \{0,1\}^n$, 使得其与真实数据库 d 几乎完全匹配, 或者说偏移量不超过 $o(n)$.

♣

如果一个算法是 BNP 的，那么这个算法之下毫无隐私可言。这就是重构攻击（reconstruction attack），随后我们可以证明出一般的方案都是 BNP 的。

定理 1.1

如果查询者被允许进行 2^n 次子集查询，并且管理者添加了 E 的噪声，那么根据响应结果，查询者可以重构出偏移量为 4E 的数据库。

♡

证明

由于查询者可以进行 2^n 次子集查询，那么他可以得到 2^n 个响应, 记响应总体为 $r(S)$
遍历所有 $c \in \{0,1\}^n$，剔除掉所有的 $|\sum c_i - r(S)| > E$，剩下的 c 就是可能的情况。
显然，真正的数据库 d 不会被剔除，由是我们考察两个集合 $I_0 = \{i|d_i = 0\}, I_1 = \{i|d_i = 1\}$

$$\begin{cases} |\sum_{i\in I_0} c_i - r(I_0)| \le E \\ |\sum_{i\in I_0} d_i - r(I_0)| \le E \end{cases}$$

由三角不等式得 $|\sum_{i\in I_0}(d_i - c_i)| \le 2E$, 对于 I_1 同理，故总偏移量为 $4E$
简要来说，这证明了该算法就是全然不隐私的。当然一个现实的问题是，查询者不可能进行指数量级的查询，这是一种低效的攻击，下面的攻击更高效，更具有现实意义。

定理 1.2 (Dinur-Nissim attack)

如果查询者被允许进行 O(n) 次子集查询，管理者加入噪声界限 $E = O(\alpha\sqrt{n})$, 那么根据响应结果，查询者可以重构出偏移量为 $O(\alpha^2)$ 的数据库。

♡

证明 具体的证明是困难的，这里只给出一些直觉上的分析

由于查询者只能进行 O(n) 次随机且均匀子集查询，仿照上文的 $c \in \{0,1\}^n$，利用线性回归剔除掉不合理的值。此处我们做一个转换，令 $c, d, S \in \{-1,+1\}^n$...

假设可能的 c(candidates) 与真实的数据库 d 在 $\Omega(n)$ 的数据上是不重合的, 考察 c 与 d 的差异到底有多大，$(c-d)\cdot S = \sum (c_i - d_i)\cdot S_i$

1. 如果 $c_i = d_i$，那么 $(c-d)\cdot S$，不影响结果。
2. 如果 $c_i \neq d_i$，那么

$$(c-d)\cdot S = \begin{cases} 2 & \text{w.p.}\frac{1}{2} \\ -2 & \text{w.p.}\frac{1}{2} \end{cases}$$

所以 $(\sum (c_i - d_i)\cdot S_i) \sim Bin(\Omega(n), \frac{1}{2})$，放缩移位后它取得 $\Omega(\sqrt{n})$ 的概率还是很高的。

由于管理者加入的噪声被限制在 $E = O(\sqrt{n})$，这使得不少 c 可以被排除。将偏移量太远的 c 排出后，我们对可能情况取一个并集，这样就十分接近真实情况了。严谨的数学证明还是参照 Dwork书里定理 8.2 的证明吧。

让我们回顾一下本小节，第一种攻击需要 2^n 次的查询以消除 $O(n)$ 的噪音, 第二种攻击需要 $\Omega(n)$ 次的查询以消除 $O(\sqrt{n})$ 的噪音. 而差分隐私允许在 $O(\sqrt{n})$ 的噪声条件下，进行 $O(n)$ 次查询，一般是通过拉普拉斯或者高斯机制去实现。其实隐私上限还是有”缩水“的空间的。比方说，我们查询的数据量远小于 n, 例如仅请求 $m << n$ 次的查询，那么相应的，管理者只需要加入 $O(\sqrt{m})$ 噪音就可以了。

正式的讨论，还是留给我们进入差分隐私的时候再说吧。

1.4 悬赏

到目前为止，所有的讨论看起来都像是在纸上谈兵。你是否在想，在现实生活中，这些攻击是不是真的可以实现呢？Aircloak 公司表示，它可以一战。

2017 年，Aircloak 公司发布了新系统 Diffix，它是一个数据库查询系统。不过，它的工作方式，看上去完全违反了上一节为了安全加以的诸多限制，首先它允许不限次数的查询，其次，它加入的噪声大小远远小于差分隐私保护机制起效所需的噪声量。尽管如此，他们还是大大咧咧的开出了 5000 美元的悬赏：给出一个攻击 Diffix 的方法，并重构数据库。

与以前类似，数据分析人员可以进行子集查询。主要的区别是，随着每次查询进行，噪声的大小呈条件集本身大小的平方根形式增加。其他防止攻击的方法包括限制计数量，调整极端值并且，禁用了"OR" 操作符。

回顾我们此前攻击做法，Dinur-Nissim 要求进行随机式地查询。这里先是获取随机的查询子集总体，然后利用一些条件集使得查询子集的特征得以显现。即便我们忽略不得使用"OR" 操作符的限制，看上去为了使得 k 个查询子集得以区分，我们似乎不得不加入 k 个条件集 (诸如此类的其他要求)。

Cohen 和 Nissim 天才式地绕过了这些限制，他们提出一个有效的新思路：相比与之前先选择查询子集再加入条件集使之凸显，他们在查询时就加入了极少量的条件。经历了精心设计的查询条件后，得到响应数据的随机性足以重构整个数据库。

具体方法解释如下。每个数据库中的用户对应唯一的 client-id, 问题是是否有一个函数以 client-id 作为参数，并将它“足够随机地”地隐含在查询集中？他们使用由四个变量指定的函数：mult、exp、d、pred。前三个变量是数字，第四个变量是一个真伪判断 (T/F),。即 $(mult * client - id)^{exp}$ 这个表达式的 d 位是否满足某个条件？它的 SQL 实现如下：

```
SELECT count(clientId)
FROM loans
WHERE floor(100 * ((clientId * 2)^0.7) + 0.5) = floor(100 * ((clientId * 2)^0.7))
AND clientId BETWEEN 2000 and 3000
AND loanStatus = 'C'
```

最后一位 loanStatus 就是他们尝试攻击的隐私位，他们攻击的用户范围是 2000 到 3000 之间的用户。如上所示，他们只是使用了一个条件，这给每次查询得到响应数据里，只带来了常数量的噪声，比上文所需的 $O(\sqrt{n})$ 噪声要远远小得多。最后，他们稍作调整除杂就 100% 地重构了整个数据库，拿走了奖金！详情参照此论文

第二章 Introduction to Differential Privacy

内容提要

- ❑ 差分隐私定义
- ❑ 相关性质

2.1 差分隐私定义

为了精准描述隐私程度，我们引入差分隐私的概念，这有时也叫中心化差分隐私模型 (central Differential Privacy) 或者置信管理者模型 (Trusted Curator)

我们假定有 n 个个体，从 $X_1 \sim X_n$, 他们将各自的数据传递给置信管理者。管理者将数据利用算法 M 输出一个公开结果。所谓差分隐私就是这个算法 M 的性质：单个个体的数据不会对算法整体输出造成太大的影响。

定义 2.1

给定算法 $M : \mathrm{X}^n \to \mathrm{Y}$, 现在我们考察 X^n 中任意两个数据集 X, X', 它们两个仅在某一条目上不一致，我们称其为“邻近数据集”。由是，我们称如下算法算法 M 是 ϵ–(纯) 差分隐私的, 如果：

对于任意邻近数据集 $X, X' \in \mathrm{X}^n$, 以及任意 $T \in \mathrm{Y}$, 都满足:

$$Pr[M(X) \in T] \leq e^{\epsilon} Pr[M(X') \in T] \tag{2.1}$$

2.2 相关性质与解释

1. ϵ 越小，意味着隐私性越强
2. 隐私性与准确性始终是矛盾的一对命题。ϵ 的取值也大有讲究，一般在在 $0.1 \sim 5$ 为宜
3. 为什么采用 e^{ϵ}? 好吧，当 ϵ 足够小的时候，你不妨考虑泰勒展开，$e^{\epsilon} \approx \epsilon + 1$, 这个看上去自然多了。当然，这种 e 指数在后续考虑“群体差分隐私”的时候还是十分便捷的，利用两个同底指数相乘化简为指数相加可以轻松化简！...

结论

让我们回顾一下：差分隐私意味着什么？简单地重复这个定义：无论单个个体包含或不包含在数据集内，算法输出结果的概率都是相近的。这意味着差分隐私可以做一些事，也说明了它所不能做的事。

首先，它可以阻止我们此前提到的诸多攻击类型。比方说 linkage attack——通过比对包含相同条目的几个数据库从而识别隐私。它还可以防止重建攻击，在某种意义上“匹配”了 Dinur-Nissim 攻击中显示的噪声边界，我们将在后续章节中对此进行量化分析。

不过，差分隐私并不妨碍对个体做出的推断。换句话说：差分隐私并不妨碍统计数据和机器学习。以经典的“吸烟会导致癌症”为例。假设一个吸烟的人正在权衡他们选择参加医学研究的选择，该研究调查吸烟是否会导致癌症。他们知道，这项研究的积极结果将对他们有害，因为它会导致他们的保险费上升。他们也知道，这项研究是通过不同的私人方式进行的，所以他们选择参与研究，他们也知道他们的隐私将会得到尊重。不幸的是，研究表明吸烟确实会导致癌症！这是一种侵犯隐私的行为，对吧？不过，差分隐私确保研究结果不会受到他们参与的显著影响。换句话说，不管他们是否参与其中，结果无论如何都会显现出来。

差异隐私也不适用于目标是识别特定个人的情况，而这与该定义是截然相反的。举个当下的例子，尽管人们强烈要求追踪 COVID-19 感染者的行踪，但目前还不清楚如何利用差异隐私来促进个人层面的接触者追踪。这需要考虑一个特定个体在哪里，以及他们与哪些特定个体互动过。另一方面，如果许多检测呈阳性的人都参加了同一项活动，那么就有可能促进总体水平的跟踪。

2.3 差分隐私的优势与局限性

根据上文分析，我们可以总结差分隐私所具有的六大优势如下:

1. 对于任意先验知识风险的防范性。在上文中我们提到，由于数据分析师的先验知识背景非常多样，因此数据分析师的辅助信息也会将一些不破坏隐私的查询变得破坏隐私。但是差分隐私与先验知识完全无关，可以抵御所有依靠辅助信息所进行的攻击。
2. 对于链式攻击的防范。将隐私攻击手段拆成多个问题，或者对数据集进行多次差分都不会泄露隐私。
3. 传递性带来的隐私闭包。由于差分隐私的传递特性，数据分析师在没有其他有关私有数据库的知识的情况下，其所做的任何进一步处理也具有差分隐私特性。也就是说，数据分析师不能仅仅通过坐在角落里思考算法的输出，得到任何会泄露个人隐私的结论。
4. 对隐私损失的量化。差分隐私利用 (ϵ,δ) 的大小对隐私损失进行度量。这就允许在不同技术之间进行比较：对于固定的隐私损失 (ϵ,δ)，哪种技术可以提供更好的准确性？为了达到固定的精度，哪种技术可以提供更好的隐私保护？
5. 对团体数据隐私的保护。差分隐私对于团体数据（例如家庭数据）所带来的隐私损失也进行了分析与控制。
6. 对于数据分析 Pipeline 的组合（Composition）隐私进行度量。一般而言，数据分析师会用多种随机算法进行分析，而不同的随机算法会带来不同的隐私损失。例如，深度学习模型训练的过程中，每一个 Epoch 都会发布一个中间模型，随着获取的中间模型越来越多，我们需要度量所有中间模型所带来的隐私损失；又例如一个可视分析系统往往有多个视图，每个视图的底层算法也不一样（如 t-sne 和热力图）。对于一个由多种随机算法组合系统如何度量隐私，以及如何设计每一个模块的 (ϵ,δ) 使得整个系统的隐私损失最小，差分隐私给出了理论思路，能够通过多个简单差分隐私模块设计和分析复杂的差分隐私算法。

但是，差分隐私不是万能的，也有其局限性。

正如上文在“吸烟引起癌症”示例中看到的，差分隐私不能保证参与人无条件不受伤害，换句话说，差异性隐私并不能保证人们认为属于自己的特性不因他人接受的调查所泄露，它只是确保不会透露自己参与了数据分析，也不会披露个体参与数据分析的任何细节。从群体调查中得出的统计结论很有可能反映出个体的统计信息，例如，通过他人数据训练的算法推荐给用户的视频往往是符合用户独特品味的，但这并不表示存在隐私权的侵害，因为用户数据甚至可能没有参加过模型训练。也就是说，差分隐私确保无论个人是否参加数据分析，都将以非常相似的概率符合统计结论性的结果。而如果分析结果告诉我们，某些特定的私有属性与公开可观察的属性密切相关，这并不违反差分隐私，因为这种相同的相关性将以几乎相同的概率被独立观察到，无论任意个体是否出现在分析所用的数据库中。

第三章 Lapalce Mechanism

内容提要

- ❏ l_1 灵敏度
- ❏ Laplace 分布
- ❏ Laplace Mechanism
- ❏ Counting queries & Histograms

3.1 基础知识

定义 3.1 (l_1 灵敏度) 我们令 $f:\mathrm{X}^n \to \mathrm{R}^k$，$X, X'$ 为邻近数据集，定义 l_1 灵敏度为：

$$\Delta(f) = max_{X,X'}||f(X) - f(X')||_1 \tag{3.1}$$

注 下面以 Δ 简代 l_1 灵敏度。差分隐私总是试图掩盖数据集里单个样本的贡献。因此，考量“函数在仅仅改变一处样本时变化的上界”看上去是十分符合直觉的

举一个简单的例子，考虑函数 $f(x) = \frac{1}{n}\sum_{i=1}^{n} X_i$, 其中 $X_i \in \{0,1\}$, 很容易推知 $\delta = \frac{1}{n}$

定理 3.1 (拉普拉斯分布) 位置参数为 0，尺度参数为 b 的拉普拉斯分布概率密度函数为

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

它的方差是 $2b^2$,(图像上相当于 x>0 时的指数分布函数沿着 y 轴对称得到)

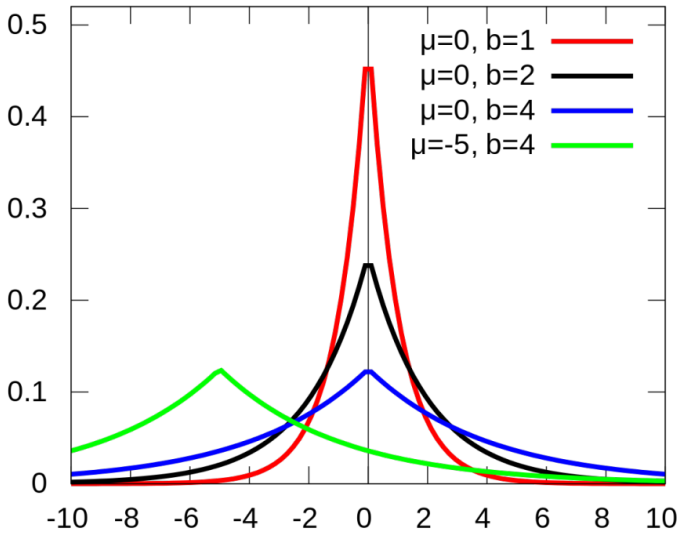


图 3.1: 拉普拉斯分布示意图

3.2 拉普拉斯机制定义

定义 3.2 (Laplace Mechanism)

令 $f:\mathrm{X}^n \to \mathrm{R}^k$, 拉普拉斯噪声机制如下：

$$M(x) = f(x) + (Y_1, Y_2, \ldots, Y_k), \text{其中 } Y_i \text{ 是属于 Laplace}(\Delta/\epsilon) \text{ 的随机变量} \tag{3.2}$$

例如 $f(x) = \frac{1}{n}\sum_{i=1}^{n} X_i$

由上文我们得到的 $\delta = \frac{1}{n}$，因此拉普拉斯机制下，$\tilde{p} = f(X) + Y$, 其中 $Y \sim Laplace(\frac{1}{n\epsilon})$

回顾前文定义的 $p = f(x)$，由于此处拉普拉斯函数位置参数为 0，所以 $E[\tilde{p}] = p$,

而 $Var[\tilde{p}] = Var[Y] = O(\frac{1}{n^2\epsilon^2})$, 由切比雪夫不等式做出估计 $|\tilde{p} - p| \leq O(\frac{1}{n\epsilon})$(在此范围内概率极高)

定理 3.2

拉普拉斯机制是 $\epsilon - DP$ 的。

证明

我们令 X, Y 邻近数据集，令 $p_X(z), p_Y(z)$ 为在某处 $z \in R^k$时$M(X), M(Y)$ 的概率密度函数，只要证明两者比值上界为 e^ϵ 即可。

$$\begin{aligned}
\frac{p_X(z)}{p_Y(z)} =& \frac{\prod_{i=1}^{n} exp(-\frac{\epsilon|f(X_i - z)|}{\Delta})}{\prod_{i=1}^{n} exp(-\frac{\epsilon|f(Y_i - z)|}{\Delta})} \\
=& \prod_{i=1}^{n} exp(-\frac{\epsilon}{\Delta}(|f(X_i - z)| - |f(Y_i - z)|)) \\
\leq& \prod_{i=1}^{n} exp(-\frac{\epsilon}{\Delta}|f(X_i) - f(Y_i|)) \\
=& exp(-\frac{\epsilon}{\Delta}\sum_{i=1}^{n}|f(X_i) - f(Y_i)|) \\
=& exp(-\frac{\epsilon}{\Delta}||f(X) - f(Y)||_1) \\
\leq& exp(\epsilon)
\end{aligned}$$

第一处不等号使用了三角不等式放缩，第二处是利用了 l_1 灵敏度的定义。

3.3 应用

3.3.1 Counting Queries

来看一下拉普拉斯机制在一些场景中的应用。我们可以问这样一个问题："数据集中有多少人具有属性 P?"与之前分析非常相似。每个个体都会有一个小的 $X_i \in \{0, 1\}$，表示它们是否具有 P，我们考虑的函数 f 是它们的和。易得，l_1 灵敏度为 1，因此该统计量 $\epsilon - DP$ 值为 $f(X) + Laplace(1/\epsilon)$。这将导致 $O(1/\epsilon)$ 的查询错误，(与数据库的大小无关)。

如果我们想回答很多查询呢？假设我们有 k 个计数查询 $f = (f_1, \ldots, f_k)$，这些都是预先指定的。我们将输出向量 $f(X)+Y$,其中 Y_i 是 i.i.d. 拉普拉斯分布的随机变量。但是我们对 Y_i 应该使用什么尺度参数呢？每个单独的计数查询f_j 的敏感性为 1, 但是我们使用相同的数据集来回答所有的查询, 因此更改单个个体可能会同时影响多个查询的结果。例如，考虑两个个体的交换：一个不满足任何属性，另一个满足任何每个属性。这个交换将使每个查询的结果改变 1, 因此总体 l_1 敏感度为 k。让我们用数学方法来分析一下。由于 $f(X) = \sum P(f1(X_i), \ldots, fk(X_i))$, 如果相邻的数据集 X 和 Y 不同，一个包含 x，另一个包含 y，那么灵敏度可以被写成 $\sum_j |f_j(x) - f_j(y)|$。它的上界可以确定：$\sum_j |f_j(x) - f_j(y)| \leq \sum_j 1 = k$。

有了这个灵敏度约束的 $\Delta = k$，我们可以在每个坐标中添加 $Y_i \sim Laplace(k/\epsilon)$ 噪声，以 $O(k/\epsilon)$ 级的误差回答每个计数查询。

总结一下。首先，这种回答 k 个计数查询的方法要求我们预先指定所有的查询——换句话说，这是一个非自适应 (non-adaptive) 的设置。我们稍后将看到，在自适应设置中，类似的保证是可以实现的，其中查询的选择可能取决于以前的查询。其次，让我们将其与之前讨论的 Dinur-Nissim 攻击进行比较。

如果查询者进行 $\Omega(n)$ 计数查询，由管理者添加 $O(\sqrt{n})$ 的噪声来防御，查询者可以重建数据库 (BNP)。

形成比较的，如果查询者进行 $O(n)$ 计数查询，并且管理者添加了 $O(n/\epsilon)$ 的噪声，那么隐私就得到了保护。

这似乎是两个结果中的一个巨大差距. 是否有更强大的攻击，让对手在更多的噪音下成功？或者我们可以减少噪音，同时保持隐私吗？幸运的是，后者是可行的，我们接下来会进一步讨论。

3.3.2 Histogram

另一种查询类型是 histogram queries。对于计数查询，改变单个个体可能会同时影响每个查询的结果。但是 histogram queries 不是这样。假设数据集中的每个人都有一些分类特征，例如，某人的年龄。我们想回答诸如“数据集中有多少人 X 岁了”这样的问题?”虽然这类似于计数查询示例，但这里每个人年龄显然只有一种。定义函数 $f : (f_0, \ldots, f_{k-1})$, 其中 f_i 问有多少人是 i 岁。这个函数的敏感度是 2：改变任何一个人的年龄都会导致一个计数减少，另一个计数增加。所以最终结果是 $f(X) + Y$, 其中 $Y_i \sim Laplace(2/\epsilon)$.

这导致了多少误差？与之前一样，我们观察到任何单个计数都会有 $O(1/\epsilon)$ 的误差。我们使用如下定理3.3来考量整体计数的误差。

定理 3.3

如果 $Y \sim Lapalce(b)$，那么

$$Pr[|Y| \geq tb] = exp(-t)$$

♡

现在，对于直方图里第 i 个 bin，计数中的误差正好是 Y_i，我们有

$$Pr[|Y_i| \geq 2\log(k/\beta)/\epsilon] \leq \beta/k$$

换句话说：误差的大小只与 bin 的数量成对数关系，而不是当我们的计数查询的线性关系

3.4 差分隐私的性质

差分隐私是一个强大的隐私保护工具，但是要实现强差分隐私约束（即 ϵ,δ→0）往往需要增加大量噪声。一般而言，数据的分析包括多个环节，如果在每个环节上都满足差分隐私约束，势必会令数据分析结果失真。差分隐私的传递性给出了一个强有力的断言：只要在数据分析的任何一个环节，随机算法 M 满足 (ϵ,0)-DP，那么在仅用该环节的结果作为输入的任意之后的数据处理过程都满足 (ϵ,0)-DP，

定理 3.4 (Post-Processing)

令 $M : X^n \to Y$ 是 $\epsilon - DP$ 的，令 $F : Y \to Z$ 为任意映射，那么 $F \circ M$ 也是 $\epsilon - DP$ 的

♡

证明

对邻进数据集中任意一对 $x, y, \|x - y\|_1 \leq 1$, 取 $S \subseteq Y$, 令 $T = \{r \in R : f(r) \subseteq S\}$

$$\begin{aligned} Pr(f(M(x)) \in S) &= Pr(M(x) \in T) \\ &\leq exp(\epsilon) Pr(M(y) \in T) \\ &= exp(\epsilon) Pr(f(M(y)) \in S) \end{aligned}$$

定理 3.5 (Group Privacy)

令 $M : X^n \to Y$ 是 $\epsilon - DP$ 的, 假定两个数据集 X, X' 在 k 个位置上的数据都不同，那么对于所有的 $T \subseteq Y$, 都有

$$Pr(M(X) \in T) \leq exp(k\epsilon)Pr(M(X') \in T)$$

♡

证明 构造 $X^{(0)} = X, \sim X^{(k)} = X'$ 上进行插值，构造一系列临近数据集即可证明

$$\begin{aligned} Pr(M(X) \in T) &= Pr(M(X^{(0)}) \in T) \\ &\leq exp(\epsilon)Pr(M(X^{(1)}) \in T) \\ &\leq exp(2\epsilon)Pr(M(X^{(2)}) \in T) \\ &\leq \dots \\ &\leq exp(k\epsilon)Pr(M(X^{(k)}) \in T) \end{aligned}$$

定理 3.6 (Composition)

$M = (M_1, M_2, \dots, M_k)$ 是一系列 $\epsilon - DP$ 函数 (选取是顺序的，自适应的)，那么最终 M 是 $k\epsilon - DP$ 的

♡

证明 考虑临近数据集 X, X', 对于函数顺序输出的 $y = (y_1, \dots, y_k)$, 我们有：

$$\begin{aligned} \frac{Pr(M(X) = y)}{Pr(M(X') = y)} &= \prod_{i=1}^{k} \frac{Pr(M_i(X) = y_i)|M_1(X) = y_1, \dots, M_i(X) = y_i}{Pr(M_i(X') = y_i)|M_1(X') = y_1, \dots, M_i(X') = y_i} \\ &\leq \prod_{i=1}^{k} exp(\epsilon) \\ &= exp(k\epsilon) \end{aligned}$$

第四章 Approximate Differential Privacy

内容提要

- ❏ $(\epsilon, \delta) - DP$
- ❏ Privacy loss
- ❏ Gausssian Mechanism

4.1 两个定义

近似差分隐私将拥有稍弱的隐私保护，但允许我们添加更少的噪音来实现 DP

定义 4.1 (近似 DP)

我们称如下算法 M 是 $(\epsilon, \delta) - DP$, 如果对于任意邻近数据集 $X, X' \in \mathrm{X}^n$, 以及任意 $T \in \mathrm{Y}$, 都满足:

$$Pr[M(X) \in T] \leq e^{\epsilon} Pr[M(X') \in T] + \delta \tag{4.1}$$

定义 4.2 (Privacy Loss)

Y,Z 为两个随机变量。隐私损失随机变量定义为 $\mathcal{L}_{Y||Z} = \ln(\frac{Pr(Y=t)}{Pr(Z=t)})$

4.2 Gaussian Mechanism

定义 4.3 (l_2 灵敏度)

我们令 $f : \mathrm{X}^n \to \mathrm{R}^k$，$X, X'$ 为邻近数据集，定义 l_2 灵敏度为:

$$\Delta_2^{(f)} = max_{X,X'} ||f(X) - f(X')||_2 \tag{4.2}$$

定义 4.4 (Gaussian distribution)

高斯分布 $N(\mu, \sigma^2)$ 的概率密度函数是:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{4.3}$$

一个有用的性质: $X, Y \sim N(0,1)(i.i.d.)$, 那么 $aX + bY \sim N(0, a^2 + b^2)$

定义 4.5 (Gaussian Mechanism)

令 $f : \mathcal{X}^n \to \mathcal{R}^k$，$\Delta_2^{(f)}$ 是 f 的 l_2 灵敏度，高斯机制 M_f 是一个随机函数，其输出是: $M(X) = f(X) + Y$, 其中 $Y_i \sim N(0, 2\ln(1.25/\delta)\Delta_2^2/\epsilon^2)$

在相同的 ϵ 下，我们可以按照与拉普拉斯机制完全相同的方法使用高斯机制，很容易对比两者效果。

```
epsilon = 1
vals_laplace = [np.random.laplace(loc=0, scale=1/epsilon) for x in range(100000)]

delta = 10e-5
sigma = np.sqrt(2 * np.log(1.25 / delta)) * 1 / epsilon
vals_gauss = [np.random.normal(loc=0, scale=sigma) for x in range(100000)]
```

```
plt.hist(vals_laplace, bins=50, label='拉普拉斯')
plt.hist(vals_gauss, bins=50, alpha=.7, label='高斯');
plt.legend();
```

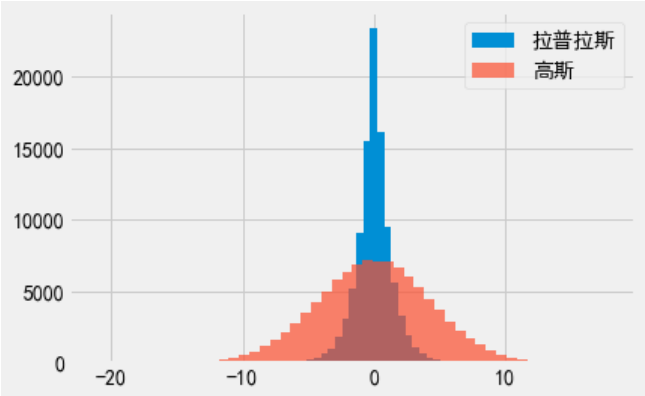


图 4.1: 拉普拉斯和高斯机制的分布

与拉普拉斯机制的曲线相比，高斯机制的曲线看起来更”平”。当应用高斯机制时，我们更有可能得到远离真实值的差分隐私输出结果，而拉普拉斯机制的输出结果与真实值更接近一些（相比之下，拉普拉斯机制的曲线看起来更”尖”）

因此，高斯机制有两个严重的缺点：其一，该机制需要使用宽松的差分隐私定义；其二，该机制的准确性不如拉普拉斯机制。既然如此，为什么我们还需要高斯机制呢？

然而，这两种机制的扩展结果间有一个关键差异点：向量值拉普拉斯机制需要使用 l_1 敏感度，而向量值高斯机制既可以使用 l_1 敏感度，也可以使用 l_2 敏感度。这是高斯机制的一个重要优势。对于 l_2 敏感度远低于 l_1 敏感度的应用来说，高斯机制添加的噪声要小得多。

定理 4.1

Gaussian Mechanism 是 $(\epsilon, \delta) - DP$ 的

♡

定理 4.2

X, X' 是 $\mathcal{X}^n$ 上的临近数据集，对于 $f : \mathcal{X}^n \to R^k$, $M(Y) = f(Y) + N(0, \sigma^2 I)$, 隐私损失

$$\mathcal{L}_{M(X)||M(X')} \sim N(\frac{\|f(X) - f(X')\|_2^2}{2\sigma^2}, \frac{\|f(X) - f(X')\|_2^2}{\sigma^2})$$

♡

4.3 近似差分隐私的性质

定理 4.3 (Post-Processing)

令 $M : X^n \to Y$ 是 $(\epsilon, \delta) - DP$ 的，令 $F : Y \to Z$ 为任意映射，那么 $F \circ M$ 也是 $(\epsilon, \delta) - DP$ 的

♡

定理 4.4 (Group privacy)

令 $M : X^n \to Y$ 是 $(\epsilon, \delta) - DP$ 的, 假定两个数据集 X, X' 在 k 个位置上的数据都不同，那么对于所有的 $T \subseteq Y$, 都有

$$Pr(M(X) \in T) \leq e^{k\epsilon} Pr(M(X') \in T) + ke^{(k-1)\epsilon}\delta$$

♡

定理 4.5 (Composition)

令 $M_1 : X^n \to Y$ 是 $(\epsilon_1, \delta_1) - DP$ 的, 令 $M_2 : Y \to Z$ 是 $(\epsilon_2, \delta_2) - DP$ 的, 那么 $M_2 \circ M_1$ 也是 $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2) - DP$ 的

4.4 高级组合性

我们已经学习了差分隐私机制的两种组合方式：串行组合性和并行组合性。事实证明，(ϵ, δ)-差分隐私引入了差分隐私机制串行组合性的一种新分析方法，此分析方法可以进一步降低隐私消耗量。

高级组合定理通常用 k-折叠适应性组合（k-fold Adaptive Composition）机制来描述。k-折叠适应性组合指的是将一系列机制组合起来，这些机制满足下述条件：

1. 适应性 adaptive：可以根据所有前述机制 $m_1, \ldots, m_{i-1}$ 的输出来选择下一个机制
2. 组合性 composition：每个机制 m_i 的输入既包括隐私数据集，也包括前述机制的所有输出

4.4.1 Advanced Composition for $\epsilon - DP$

对比一下，之前的串行组合性说总隐私消耗量为 $k\epsilon$, 而高级组合性称：

- 如果 k 适应性组合 $m_1, \ldots, m_k$ 中每一个机制 m_i 都满足 $\epsilon - DP$
- 则对于任意 δ，整个 k-折叠适应性组合满足 (ϵ', δ')-DP，其中

$$\epsilon' = 2\epsilon\sqrt{2k\log(1/\delta')}$$

对于相同机制，应用高级组合性得到的 ϵ' 下界远低于应用串行组合性得到的下界。这意味着什么呢？这意味着串行组合性得到的隐私消耗量下界是宽松的。与实际隐私消耗相比，串行组合性得到的下界不够紧致。事实上，高级组合性得到的下界也是宽松的，此下界只是比串行组合性给出的下界稍显紧致一些。

需要着重强调的是，得到的两种隐私消耗量上界在技术角度看不具有可比性，因为高级组合性引入了 δ。但当 δ 很小时，我们通常可以比较两种方法给出的 ϵ

事实证明，当 k 小于 70 时，标准的串行组合性比高级组合性得到的总隐私消耗量更小。因此，仅当 k 比较大时（如大于 100 时），高级组合性才会有用武之地。不过，当非常大时，高级组合性可以显露出巨大的优势。

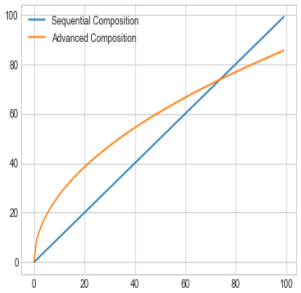


图 4.2: small k

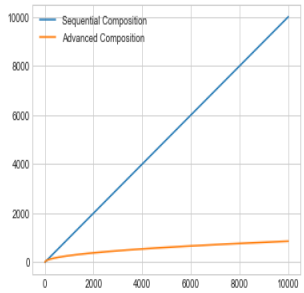


图 4.3: large k

4.4.2 Advanced Composition for $(\epsilon, \delta) - DP$

高级组合性更加一般的描述如下

- 如果 k-折叠适应性组合 $m_1, \ldots, m_k$ 中的每个机制 m_i 都满足 $(\epsilon, \delta) - DP$
- 则对于任意 $\delta' \geq 0$，整个 k-折叠适应性组合满足 $(\epsilon, k\delta + \delta')$-DP, 其中

$$\epsilon' = 2\epsilon\sqrt{2k\log(1/\delta')}$$

第五章 Local Sensitivity

内容提要

- ❏ Global & Local Sensitivity
- ❏ 使用”建议-测试-发布”框架来安全地使用局部敏感度
- ❏ 平滑敏感度框架
- ❏ 使用”采样-聚合”框架来回复敏感度为任意值的问询

5.1 Global & Local Sensitivity

截至目前，我们只学习了一种敏感度指标：全局敏感度。全局敏感度定义考察的是任意两个临近数据集。这种定义似乎显得过于严苛了。由于我们将在实际数据集上执行差分隐私机制，我们难道不应该只需要考虑此数据集的临近数据集吗？

局部敏感度（Local Sensitivity）的直观思想是：将两个数据集中的一个固定为待问询的实际数据集，仅考虑此数据集的所有临近数据集。

定义 5.1 (Local Sensitivity)

函数 $f : \mathcal{D} \to \mathcal{R}$ 局部敏感度定义如下：

$$LS(f, x) = \max_{x' : d(x,x') \leq 1} |f(x) - f(x')|$$

♣

注意到，局部敏感度是以问询（f）和实际数据集（x）这两个输入定义的函数。与全局敏感度不同，我们不能抛开输入的数据集而单独讨论局部敏感度。反之，我们需要考虑局部敏感度所依赖的实际数据集是什么

5.1.1 均值问询的局部敏感度

很难为一些函数设置全局敏感度的上界。此时，局部敏感度就可以被派上用场，以便我们对这些函数设置有界的敏感度。均值函数就是一个典型的例子。截至目前，为了使均值问询满足差分隐私，我们需要将均值问询拆分为两个问询：满足差分隐私的求和问询（分子）和满足差分隐私的计数问询（分母）。应用串行组合性和后处理性，这两个问询结果的商仍然满足差分隐私。

我们为什么非要通过这种方式来回复均值问询呢？因为均值问询的输出结果依赖于数据集的大小。从数据集中增加或删除数据行，数据集的大小将随之变化，导致均值问询的输出结果发生变化。如果我们想计算均值问询的全局敏感度上界，我们就需要假设可能出现的最糟糕情况：数据集的大小为 1。如果数据属性值的上下界分别为 u 和 l，则均值的全局敏感度为 $|u - l|$。对于较大规模的数据集来说，此全局敏感度上界是极为严苛的。与之相比，”噪声求和除以噪声计数”的问询回复方法要好得多

局部敏感度定义下的情况就有所不同了。我们考虑最糟糕的情况：添加一个包含最大值 u 的新数据行。令 $n = |x|$（即表示数据集的大小）。我们先考虑实际数据集的均值：

$$f(x) = \frac{\sum_{i=1}^{n} x_i}{n}$$

我们看看向其中添加一行后会发生什么？

$$\begin{aligned} |f(x') - f(x)| &= \left| \frac{\sum_{i=1}^{n} x_i + u}{n} - \frac{\sum_{i=1}^{n} x_i}{n} \right| \\ &\leq \left| \frac{\sum_{i=1}^{n} x_i + u}{n} - \frac{\sum_{i=1}^{n} x_i}{n+1} \right| \\ &= \left| \frac{u}{n+1} \right| \end{aligned}$$

此问询的局部敏感度依赖于实际数据集的大小，而全局敏感度的定义不可能与数据集本身相关。

5.1.2 通过局部敏感度实现差分隐私？

我们已经定义了一种新的敏感度指标，但我们该如何使用它呢？我们可以像全局敏感度那样直接使用拉普拉斯机制吗？以下对 F 的定义满足差分隐私性吗？

$$F(x) = f(x) + Laplace(\frac{LS(f,x)}{\epsilon})$$

很不幸，答案是否定的。由于 LS(f,x) 本身与数据集相关，如果分析者知道某个问询在特定数据集下的局部敏感度，那么分析者也许能够推断出一些与数据集相关的信息。因此，不可能直接使用局部敏感度来满足差分隐私。

即使局部敏感度的取值对分析者保密也无济于事。分析者通过观察少量问询的回复就可能确定噪声尺度，从而使用该值推断出局部敏感度。差分隐私旨在保护的输出，但并不能保护差分隐私定义中所使用的敏感度指标。

学者们已经提出了几种安全使用局部敏感度的方法。我们将在本章剩余部分展开讨论。

辅助数据可以向我们透露出一些非常敏感的信息。试想一下，如果我们的问询是:”数据集中名叫 Joe 的人的平均成绩排名位于班级前 2% 吗?”，用于计算平均值的数据集大小就变得非常敏感了！

5.2 Propose-Test-Release Frame

5.2.1 PTR 概念

局部敏感度的主要问题是敏感度本身会泄露数据的一些信息。如果我们让敏感度本身也满足差分隐私呢？直接实现这一目标很有挑战，因为一个函数局部敏感度的全局敏感度是无上界的。不过，我们可以通过提交满足差分隐私的问询来间接得到某个函数的局部敏感度。

“建议-测试-发布”（Propose-Test-Release）框架采用的就是此种方法。该框架首先询问数据分析者函数的建议（Propose）局部敏感度上界。随后，该框架执行满足差分隐私的测试（Test），检验所问询的数据集是否远离了局部敏感度高于建议边界的数据集。如果测试通过，该框架发布（Release）噪声结果，并将噪声量校准到建议的边界。

为了回答一个数据集是否”远离”了有着更高局部敏感度数据集的问题，我们定义 k 距离局部敏感度的概念。我们用 A(f,x,k) 表示通过从数据集 x 执行 k 步可得到 f 的最大局部敏感度。用数学语言描述，我们有

$$A(f,x,k) = \sum_{y:d(x,y)\leq k} LS(f,y)$$

现在，我们准备定义一个问询来回答以下问题:”最少要多少步才能实现比给定上界 b 更大的局部敏感度?”

$$D(f,x,k) = argmin_k A(f.x.k) > b$$

最后我们定义‘Propose-Test-Release’框架的, 其满足 $(\epsilon,\delta) - DP$

定义 5.2 (PTR frame)

1. Propose: 问询建议的局部敏感度上界 b
2. Test: 如果 $D(f,x,b) + Laplace(1/\epsilon) > \frac{\log(2/\delta)}{2\epsilon}$, 返回无效
3. Release: 否则，返回 $f(x) + Laplace(b/\epsilon)$

♣

注意到的全局敏感度为 1：向添加或移除一行都可能导致距离变化为到比当前局部敏感度”高”了 1。因此，添加尺度为 $1/\epsilon$ 的拉普拉斯噪声可以得到一种度量局部敏感度的差分隐私方法。

5.2.2 代码实现

让我们来实现均值问询的”建议-测试-发布”框架吧。回想一下，该问询的局部敏感度是 $|\frac{u}{n+1}|$；提高此局部敏感度的最好方法是减小 n 。如果我们以数据集 x 为出发点执行 k 步，得到的局部敏感度就会变为 $|\frac{u}{(n-k)+1}|$

```
import pandas as pd
adult = pd.read_csv('https://github.com/jnear/cs211-data-privacy/raw/master/homework/adult_with_pii.
    csv')

def ls_at_distance(df, u, k):
  return np.abs(u/(len(df) - k + 1))

def dist_to_high_ls(df, u, b):
  k = 0
  while ls_at_distance(df, u, k) < b:
     k += 1
  return k

def laplace_mech(x, b, epsilon):
  return x + np.random.laplace(0, b/epsilon)

def ptr_avg(df, u, b, epsilon, delta, logging=False):
  df_clipped = df.clip(upper=u)
  k = dist_to_high_ls(df_clipped, u, b)

  noisy_distance = laplace_mech(k, 1, epsilon)
  threshold = np.log(2/delta)/(2*epsilon)

  if logging:
     print(f"噪声距离为{noisy_distance}，而门限值为{threshold}")

  if noisy_distance >= threshold:
     return laplace_mech(df_clipped.mean(), b, epsilon)
  else:
     return None

df = adult['Age']
u = 100                 # 设置年龄的上界为100
epsilon = 1             # 设置  = 1
delta = 1/(len(df)**2) # 设置  = 1/n^2
b = 0.005               # 建议敏感度为0.005

ptr_avg(df, u, b, epsilon, delta, logging=True)
```

Result:

- 噪声距离为 38.581701254137585，而门限值为 10.73744412245554
- 38.581701254137585

不过，局部敏感度并不总优于全局敏感度。对于均值问询，我们用旧回复策略得到的回复效果一般会好得多。这是因为我们可以将均值问询拆分为两个独立的、全局敏感度均有界的问询（求和与计数）。我们同样可以应用全局敏感度实现均值问询。

```
def gs_avg(df, u, epsilon):
df_clipped = df.clip(upper=u)

noisy_sum = laplace_mech(df_clipped.sum(), u, .5*epsilon)
noisy_count = laplace_mech(len(df_clipped), 1, .5*epsilon)

return noisy_sum / noisy_count

gs_avg(adult['Age'], u, epsilon)
```

Result:38.5866467550152

```
gs_results = [pct_error(np.mean(adult['Age'])
gs_avg(df, u, epsilon)) for i in range(100)]
ptr_results = [pct_error(np.mean(adult['Age'])
ptr_avg(df, u, b, epsilon, delta)) for i in range(100)]

_, bins, _ = plt.hist(gs_results, label='全局敏感度')
plt.hist(ptr_results, alpha=.7, label='"建议-测试-发布"框架', bins=bins)
plt.xlabel('误差率')
plt.ylabel('尝试次数')
plt.legend()
```

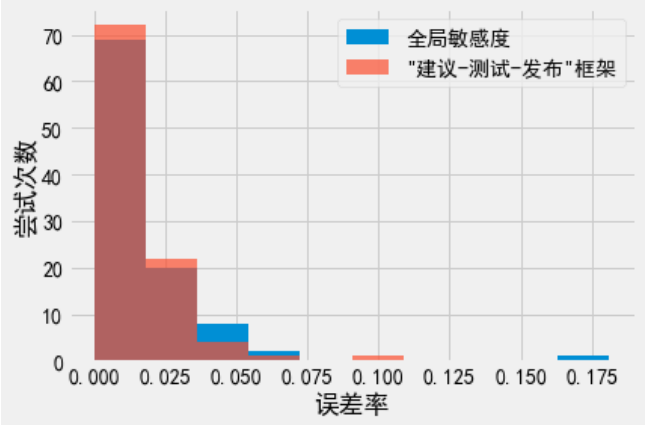


图 5.1: 全局敏感度与建议-测试-发布框架的误差率比较

5.3 平滑敏感度

第二种使用局部敏感度的方法称为平滑敏感度（Smooth Sensitivity），来自 Nissim、Raskhodnikova 和 Smith 的论文。利用拉普拉斯噪声实例化得到的平滑敏感度框架可提供 $(\epsilon, \delta) - DP$:

1. 设置 $\beta = \frac{\epsilon}{2\log(2/\delta)}$
2. 令 $S = \max_{i=1\ldots n} e^{-\beta k} A(f, x, k)$
3. 发布 $f(x) + Laplace(\frac{2S}{\epsilon})$

平滑敏感度的基本思想是不使用局部敏感度本身，而是使用局部敏感度的”平滑”近似值来校准噪声。使用平滑量的目的就是要防止因直接使用局部敏感度而意外发布数据集的有关信息。上述步骤 2 就是在执行平滑操作：利用临近数据集与实际数据集距离的指数函数来缩放局部敏感度，并取缩放程度最大的结果作为最终的

局部敏感度。这样做的效果是，如果 x 的临近数据集存在局部敏感度峰值，那么该峰值将作用于 x 的平滑敏感度中（因此，峰值本身被”平滑”了，不会泄露数据集的任何信息）。

与”建议-测试-发布”框架相比，平滑敏感度拥有明显的优势：它不需要分析者建议敏感度边界。站在分析者的角度看，使用平滑敏感度和使用全局敏感度一样简单。但是，平滑敏感度有两个主要的缺点。第一，平滑敏感度通常比局部敏感度大（至少为 2 倍，详见步骤 3），因此增加的噪声量可能会比”建议-测试-发布”等替代框架更大。第二，计算平滑敏感度时需要找到所有可能的 k 中最大的平滑敏感度，这可能涉及极大的计算开销。在多数情况下，可以证明只需要考虑少量的 k 值就足够了（对于多数问询函数，$e^{-\beta k}$ 的指数衰减效果会很快覆盖 $A(f,x,k)$ 的增长效果）。然而，对于想使用平滑敏感度的每一个问询函数，我们都需要证明此函数只需要考虑少量的值。

5.4 采样-聚合框架

我们接下来考虑与局部敏感度相关的最后一个框架，即”采样-聚合”（Sample and Aggregate）框架（同样来自 Nissim、Raskhodnikova 和 Smith 的论文）。令裁剪上界和下界分别为 u 和 l，则下述框架满足 $\epsilon - DP$

1. 将数据集 $\mathcal{X}$ 拆分为 k 个互不相交的数据块 $x_1, \ldots, x_k$
2. 计算每个数据块的裁剪回复值：$a_i = max(l, min(u, f(x_i)))$
3. 计算平均值并添加噪声：$\frac{1}{k}\sum_{i=1}^{k} a_i + Laplace(\frac{u-l}{k\epsilon})$

注意，该框架满足纯 $\epsilon - DP$，且实际执行时无需使用局部敏感度。事实上，我们不需要知道关于 f（无论是全局还是局部）敏感度的任何信息。除了知道每个数据块 x_i 互不相交外，我们也不需要知道 x_i 的任何其他信息。我们一般需要对数据集进行随机拆分（”好”的随机拆分结果往往会给出更准确的回复），但随机拆分并不是”采样-聚合”框架得以应用的必要条件。

仅仅利用全局敏感度和并行组合就可以证明该框架满足差分隐私。我们将数据集拆分为个互不相交的数据块，因此每个个体仅出现在一个数据块中。我们不知道 f 的敏感度，但我们将其输出裁剪到 u 和 l 的范围内。因此，每个裁剪回复值 $f(x_i)$ 的敏感度为 $u - l$. 由于我们调用了 k 次 f，并取 k 次回复的平均值，因此均值的全局敏感度为 $\frac{u-l}{k}$ 。

请注意，我们在”采样-聚合”框架中直接声明了均值的全局敏感度边界，并没有将均值拆分为求和问询与计数问询。我们无法对常规均值问询执行此操作，因为常规均值问询中计算平均数的分母与数据集大小相关。在”采样-聚合”框架中，计算平均数的分母由分析者所选择的 k 确定，k 的取值与数据集无关。当均值问询中计算平均数的分母可以独立确定并对外公开时，我们就可以放心地使用这一改进的全局敏感度边界。

在该”采样-聚合”框架的简单实例中，我们要求分析者提供每个 $f(x_i)$ 输出的上界和下界。由于 u 和 l 可能依赖于 f 的定义，因此可能极难确定 u 和 l 的取值。例如，在计数问询中，f 的输出与数据集直接相关。

学者们已经提出了更高级的”采样-聚合”框架实例化方法（Nissim、Raskhodnikova 和 Smith 在论文中讨论了一部分实例化方法），通过利用局部敏感度避免分析者给出和。然而，很容易限制一些特定函数的输出范围，这种情况下就可以直接使用”采样-聚合”框架。我们仍然以计算给定数据集的平均年龄为例。人口的平均年龄很大可能在 20 到 80 之间，因此设置 l=20 和 u=80 是合理的。这样一来，我们限制了数据集平均年龄问询的输出范围，从而可以直接使用”采样-聚合”框架。只要每个数据块都能体现人口信息的群体特性，不会出现过于极端的情况，我们就可以放心大胆地限制输出范围，而不丢失过多的信息。

```
def f(df):
  return df.mean()

def saa_avg_age(k, epsilon, logging=False):
    df = adult['Age']

    # 计算每个数据块应包含的行数
    chunk_size = int(np.ceil(df.shape[0] / k))
```

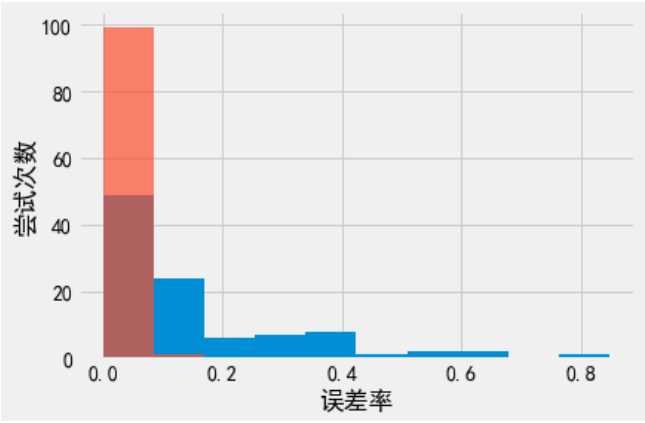
```
    if logging:
        print(f'数据块大小: {chunk_size}')
    # 步骤1: 将`df`拆分为数据块
    xs     = [df[i:i+chunk_size] for i in range(0,df.shape[0],chunk_size)]
    # 步骤2: 在每个x_i上执行f, 并裁剪输出值
    answers = [f(x_i) for x_i in xs]
    u = 80
    l = 20
    clipped_answers = np.clip(answers, l, u)
    # 步骤3: 计算输出均值, 并增加噪声
    noisy_mean = laplace_mech(np.mean(clipped_answers), (u-l)/k, epsilon)
    return noisy_mean

saa_avg_age(600, 1, logging=True)
```

该框架的关键参数是数据块数量，即 k 的取值。一方面，k 越大，噪声均值的敏感度就越小。因此数据块数量越多，噪声量越小。另一方面，k 越大，每个数据块就越小，因此每个回复值 $f(x_i)$ 都越可能远离正确回复值 $f(X)$。在上述例子中，我们希望每个数据块的平均年龄接近整个数据集的平均年龄。如果每个块只包含极少部分人，数据块的平均年龄很可能与数据集的平均年龄相差甚远。

我们应该如何设置 k 的值呢？这依赖于 f 和数据集本身，因此很难为数据集设置适当的 k 值。让我们尝试使用不同的 k 值来回复均值问询。

```
def plot_results(k):
  df = adult['Age']
  _, bins, _ = plt.hist([pct_error(np.mean(df), saa_avg_age(k, epsilon)) for i in range(100)])
  plt.hist([pct_error(np.mean(df), gs_avg(df, u, epsilon)) for i in range(100)], alpha=.7, bins=bins)
# k = 6000; "采样-聚合"框架的回复结果接近全局敏感度的回复结果了!
plot_results(6000)
plt.xlabel('误差率')
plt.ylabel('尝试次数')
```



因此，尽管”采样-聚合”框架的准确性无法击败全局敏感度方法，但如果能选择合适的 kl，两者的回复效果也可以非常接近。”采样-聚合”框架最大的优势在于此框架适用于任意函数 f。无论函数的敏感度是多少，都可以使用”采样-聚合”框架。这意味着只要 f 本身表现良好，就可以应用”采样-聚合”框架获得满足差分隐私的输出，并得到较好的准确度。另一方面，”采样-聚合”框架要求分析者设置裁剪边界 u 和 l，并设置数据块数量 k.

第六章 Expontional Mechanism

6.1 指数机制基本概念

之前我们已学习的基本机制（拉普拉斯机制和高斯机制）针对的都是数值型回复 (numerical)，只需直接在回复的数值结果上增加噪声即可。如果我们想返回一个准确结果（即不能直接在结果上增加噪声），同时还要保证回复过程满足差分隐私，该怎么办呢？

一种解决方法是使用指数机制（Exponential Mechanism）。此机制可以从备选回复集合中选出”最佳”回复的同时，保证回复过程满足差分隐私。分析者需要定义一个备选回复集合。同时，分析者需要指定一个评分函数（Scoring Function），此评分函数输出备选回复集合中每个回复的分数。分数最高的回复就是最佳回复。指数机制通过返回分数近似最大的回复来实现差分隐私保护。换言之，为了使回复过程满足差分隐私，指数机制返回结果所对应的分数可能不是备选回复集合中分数最高的那个结果。

指数机制满足 $\epsilon - DP$：

1. 选择一个备选回复集合 $\mathcal{R}$
2. 指定一个的评分函数 $u : \mathcal{D} \times \mathcal{R} \to \mathbb{R}$, 其全局敏感度为 Δ_u
3. 指数机制的输出 $r \in \mathcal{R}$，各个回复的输出概率与下述表达式成正比 $exp(\frac{\epsilon u(x,r)}{2\Delta_u})$

和我们之前学习过的机制相比，指数机制最大的不同点在于其总会输出集合 $\mathcal{R}$ 中的一个元素。当必须从一个有限集合中选择输出结果，或不能直接在结果上增加噪声时，指数机制就会变得非常有用。例如，假设我们要为一个大型会议敲定一个日期。为此，我们获得了每个参会者的日程表。我们想选择一个与尽可能少的参会者有时间冲突的日期来举办会议，同时想通过差分隐私为所有参会者的日程信息提供隐私保护。在这个场景下，在举办日期上增加噪声没有太大意义，增加噪声可能会使日期从星期五变成星期六，使冲突参会者的数量显著增加。应用指数机制就可以完美解决此类问题：既不需要在日期上增加噪声，又可以实现差分隐私。

还有几点有待讨论：

1. 无论备选输出集大小，指数机制的隐私消耗量仍然为 ϵ。我们后续将详细讨论这一点。
2. 无论是有限集合还是无限集合，均可应用指数机制。但如果是无限集合，则我们会面临一个非常有挑战的问题：如何构造一个实际可用的实现方法，其可以遵循适当的概率分布从无限集合中采样得到输出结果。

6.2 指数机制代码实现

```
options = adult['Marital Status'].unique()

def score(data,options):
    return data.value_counts()[options]/1000

score(adult['Marital Status'],'Never-married')
```

results:10.683

```
def exponential(x,R,u,sensitivity,epsilon):
  #计算R中每个回复的分数
  scores = [u(x,r) for r in R]
  #根据分数计算每个回复的输出概率
  probabilities = [np.exp(epsilon*score /(2*sensitivity)) for score in scores]
  #使得概率和为1
  probabilities = probabilities/np.linalg.norm(probabilities,ord = 1)
```

```
  #根据概率分布选择回复
  return np.random.choice(R,1,p = probabilities)[0]

exponential(adult['Marital Status'], options, score, 1, 1)
```

results:'Married-civ-spouse'

6.3 Report noisy max

我们能用拉普拉斯机制实现指数机制吗？当为有限集合时，指数机制的基本思想是使从集合中选择元素的过程满足差分隐私。我们可以应用拉普拉斯机制给出此基本思想的一种朴素实现方法：

1. 对于每个备选数据集 $\mathcal{R}$ 里每个 $r \in \mathcal{R}$, 计算噪声分数 (noisy score):$u(x,r) + Lap(\frac{\Delta_u}{\epsilon})$
2. 仅输出噪声分数最大的元素 r

因为评分函数 u(x,r) 在 x 下的敏感度为 Δ_u，所以步骤 1 中的每次”问询”都满足 $\epsilon - DP$。因此，如果包含 n 个元素，根据串行组合性，上述算法满足 $n\epsilon - DP$.

然而，如果我们使用指数机制，则总隐私消耗量将只有 ϵ！为什么指数机制效果如此之好？原因是指数机制泄露的信息更少。

对于上述定义的拉普拉斯机制实现方法，我们的隐私消耗量分析过程是非常严苛的。实际上，步骤 1 中计算整个集合噪声分数的过程满足 $\epsilon - DP$，因此我们可以发布得到的所有噪声分数。我们应用后处理性得到步骤 2 的输出满足 $n\epsilon - DP$。

与之相比，指数机制仅发布最大噪声分数所对应的元素，但不发布最大噪声分数本身，也不会发布其他元素的噪声分数。

上述定义的算法通常被称为报告噪声最大值（Report Noisy Max）算法。实际上，因为此算法只发布最大噪声分数所对应的回复，所以无论集合包含多少个备选回复，此算法都满足 $\epsilon - DP$

输出噪声最大值算法的实现方法非常简单，而且很容易看出，此算法得到的回复结果与之前我们实现的有限集合指数机制非常相似。

```
def report_noisy_max(x, R, u, sensitivity, epsilon):
  # 计算R中每个回复的分数
  scores = [u(x, r) for r in R]
  # 为每个分数增加噪声
  noisy_scores = [laplace_mech(score, sensitivity, epsilon) for score in scores]
  # 找到最大分数对应的回复索引号
  max_idx = np.argmax(noisy_scores)
  # 返回此索引号对应的回复
  return R[max_idx]

report_noisy_max(adult['Marital Status'], options, score, 1, 1)
```

因此，当 $\mathcal{R}$ 为有限集合时，可以用报告噪声最大值机制代替指数机制。但如果 $\mathcal{R}$ 为无限集合呢？我们无法简单地为无限集合中每一个元素对应的分数增加拉普拉斯噪声。当 $\mathcal{R}$ 为无限集合时，我们不得不使用指数机制。

第七章 The Sparse vector Technique

稀疏向量技术（Sparse Vector Technique，SVT）可以非常有效地节省隐私消耗量。稀疏向量技术适用于在数据集上执行敏感度为 1 的问询流。此技术只发布问询流中第一个通过测试的问询索引号，而不发布其他任何信息。稀疏向量技术的优势在于，无论总共收到了多少问询，此机制消耗的总隐私消耗量都是固定的。

7.1 AboveThreshold algorithm

算法输入是敏感度为 1 的问询流、数据集 D、阈值 T 、以及隐私参数 ϵ；算法满足 $\epsilon - DP$。

```
import random

# preserves epsilon-differential privacy
def above_threshold(queries, df, T, epsilon):
    T_hat = T + np.random.laplace(loc=0, scale = 2/epsilon)
    for idx, q in enumerate(queries):
        nu_i = np.random.laplace(loc=0, scale = 4/epsilon)
        if q(df) + nu_i >= T_hat:
            return idx
    # if the algorithm "fails", return a random index
    # more convenient in certain use cases
    return random.randint(0,len(queries)-1)

def above_threshold_fail_signal(queries, df, T, epsilon):
    T_hat = T + np.random.laplace(loc=0, scale = 2/epsilon)
    for idx, q in enumerate(queries):
        nu_i = np.random.laplace(loc=0, scale = 4/epsilon)
        if q(df) + nu_i >= T_hat:
            return idx
    # return an invalid index as the special "fail" signal
    # this is convenient for implementing the sparse algorithm
    return None
```

AboveThreshold（近似）返回 queries 中回复结果超过阈值的第一个问询所对应的索引号。算法之所以满足差分隐私，是因为算法有可能返回错误的索引号，索引号对应的问询回复结果有可能未超过给定阈值，索引号对应的问询有可能不是第一个回复结果超过阈值的问询。

算法的工作原理如下。首先，算法生成一个噪声阈值 T_hat；随后，算法比较噪声问询回复 $(q(i) + nu_i)$ 与噪声阈值；最后，算法返回第一个噪声问询回复大于噪声阈值的问询索引号。

尽管此机制会计算多个问询的回复结果，但该算法的隐私消耗量仅为 ϵ 。该算法一种可能的朴素实现方法是，先计算所有问询的噪声回复结果，再选择高于阈值的第一个问询索引号

```
# preserves |queries|*epsilon-differential privacy
def naive_above_threshold(queries, df, T, epsilon):
    for idx, q in enumerate(queries):
        nu_i = np.random.laplace(loc=0, scale = 1/epsilon)
        if q(df) + nu_i >= T:
            return idx
    return None
```

当问询总数量为 n 时，根据串行组合性，该朴素实现可以满足 $n\epsilon - DP$。

为什么 AboveThreshold 的效果会如此好呢？正如我们在指数机制中所看到的那样，串行组合性允许 AboveThreshold 发布比实际所需更多的信息。特别地，该算法的朴素实现允许发布每一个（而不仅仅是第一个）超过阈值的问询索引号，也允许额外发布噪声问询回复本身。即便额外发布了这么多信息，朴素实现依然可以满足 $n\epsilon - DP$ -差分隐私。AboveThreshold 可以隐瞒所有这些额外的信息，从而得到更加紧致的隐私消耗量。

7.2 应用稀疏向量技术

当我们想执行很多不同的问询，但我们只关心其中一个问询（或一小部分问询）的回复结果时，稀疏向量技术就有很大的用武之地了。实际上，之所以叫稀疏向量技术，正是因为此技术的适用场景：问询向量越稀疏（即大多数回复结果不会超过阈值），此技术作用最大。

在前面提到的场景中，我们已经有了一个完美的适用场景：选择求和问询的裁剪边界。之前，我们实现的方法类似于 AboveThreshold 的朴素实现：获得多个不同的裁剪边界后，分别计算噪声裁剪边界，并选择一个尽可能低的，且不会导致最终回复结果改变太大的一个裁剪边界。

我们可以通过使用稀疏向量技术获得更好的效果。考虑这样一个问询：此问询首先对数据集中每个人的年龄进行裁剪，再把裁剪结果求和：

```
 def age_sum_query(df, b):
 return df['Age'].clip(lower=0, upper=b).sum()

age_sum_query(adult, 30)
```

results:913809

我们想选择一个较好的 b。朴素算法是获取多个满足差分隐私的 b，返回使求和结果不再增大的最小的 b。

```
def naive_select_b(query, df, epsilon):
  bs = range(1, 1000, 10)
  best = 0
  threshold = 10
  epsilon_i = epsilon / len(bs)

  for b in bs:
     r = laplace_mech(query(df, b), b, epsilon_i)

     # 如果新的求和结果与旧的求和结果很接近，则停止
     if r - best <= threshold:
        return b
     # 否则，将"最佳"求和结果更新为当前求和结果
     else:
        best = r

  return bs[-1]

naive_select_b(age_sum_query, adult, 1)
```

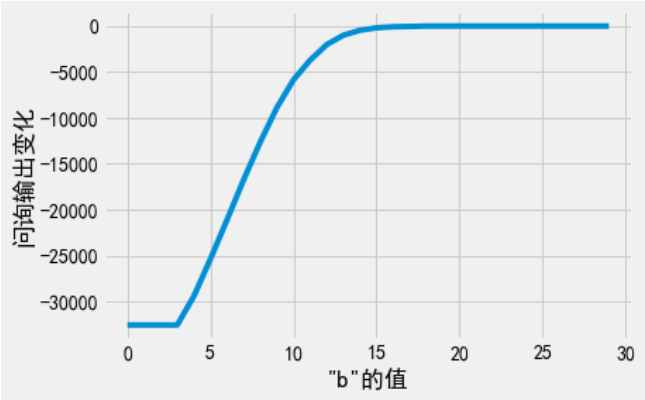
result:121

我们可以在这里使用稀疏向量技术吗？我们只关心一件事：当 age_sum_query(df, b) 停止增加时 b 的值。

然而，age_sum_query(df, b) 的敏感度就等于 b，因为增加或移除 df 中的一列会至多使求和结果改变 b；要想使用稀疏向量技术，我们需要构建敏感度为 1 的问询流。

实际上,我们真正关心的是求和结果在特定的取值下是否会变化(即 age_sum_query(df, b)−age_sum_query(df, b+1) 是否足够小)。考虑一下，如果我们向 df 增加一行数据会发生什么：问询中第一部分 age_sum_query(df, b) 的结果会增加 b，但问询中第二部分 age_sum_query(df, b+1) 的结果也会增加，只不过增加了 b+1。因此，敏感度实际上为 $|b-(b+1)|=1$。这意味着此问询的敏感度为 1，满足要求！随着 b 的值趋于最优值，问询中两部分的差值将趋近于 0：

```
bs = range(1,150,5)
query_results = [age_sum_query(adult, b) - age_sum_query(adult, b + 1) for b in bs]
plt.xlabel('"b"的值')
plt.ylabel('问询输出变化')
plt.plot(query_results);
```



根据这一观察结论，我们来定义一个求和差问询流，并基于稀疏向量技术，应用 AboveThreshold 确定最佳 b 的问询索引号。

```
def create_query(b):
  return lambda df: age_sum_query(df, b) - age_sum_query(df, b + 1)

bs = range(1,150,5)
queries = [create_query(b) for b in bs]
epsilon = .1

bs[above_threshold(queries, adult, 0, epsilon)]
```

请注意，备选 b 的列表有多长并不重要。无论此列表有多长，我们都能获得准确的结果（并消耗相同的隐私预算）。稀疏向量技术真正的强大之处在于，其消除了隐私消耗量与所执行问询数量的依赖关系。机制的输出结果不依赖于 b 的数量。即使备选 b 的列表中包含上千个元素，我们仍将得到准确的结果！

我们可以使用稀疏向量技术构建可自动计算裁剪参数的求和问询算法（也可以构建对应的均值问询算法）。

```
def auto_avg(df, epsilon):
  def create_query(b):
      return lambda df: df.clip(lower=0, upper=b).sum() - df.clip(lower=0, upper=b+1).sum()

  # 构造问询流
  bs = range(1,150000,5)
  queries = [create_query(b) for b in bs]

  # 使用1/3的隐私预算执行AboveThreshold，得到一个好的裁剪参数
  epsilon_svt = epsilon / 3
```

```
  final_b = bs[above_threshold(queries, df, 0, epsilon_svt)]

  # 分别使用1/3的隐私预算来获得噪声求和值与噪声计数值
  epsilon_sum = epsilon / 3
  epsilon_count = epsilon / 3

  noisy_sum = laplace_mech(df.clip(lower=0, upper=final_b).sum(), final_b, epsilon_sum)
  noisy_count = laplace_mech(len(df), 1, epsilon_count)

  return noisy_sum/noisy_count

auto_avg(adult['Age'], 1)
```

7.3 返回多个回复

在上述应用场景中，我们只需要得到第一个超过阈值的问询索引号。但在其他的一些应用场景中，我们可能想要找到所有超过阈值的问询索引号。

我们也可以使用稀疏向量技术实现这一点，但代价是我们必须消耗更高的隐私预算。我们可以实现 sparse（稀疏）算法来完成该任务：

1. 从问询流 $qs = [q_1, \ldots, q_k]$ 开始
2. 在问询流 qs 上执行 AboveThreshold，得到第一个超过阈值的问询索引号 i
3. 使用问询流剩余部分 $qs = [q_i, \ldots, q_k]$ 重启算法（回到步骤 1）

如果算法调用 n 次 AboveThreshold，每次调用的隐私参数为 $n\epsilon$，则根据串行组合性，此算法满足 $n\epsilon$-差分隐私。如果想在给定的总隐私消耗量下执行算法，我们就需要限制 n 的大小。也就是说，sparse 算法可以要求分析者最多调用 c 次 AboveThreshold

```
def sparse(queries, df, c, T, epsilon):
  idxs = []
  pos = 0
  epsilon_i = epsilon / c

  # 如果我们执行完问询流中的所有问询，或者我们找到了c个超过阈值的问询回复，则停止
  while pos < len(queries) and len(idxs) < c:
      # 执行AboveThreshold，寻找下一个超过阈值的问询回复
      next_idx = above_threshold_fail_signal(queries[pos:], df, T, epsilon_i)
      # 如果AboveThreshold执行完了最后一个问询，则返回所有超过阈值的问询索引号
      if next_idx == None:
          return idxs
      # 否则，更新pos，使其指向问询流中剩余的问询
      pos = next_idx+pos
      # 更新idxs，添加AboveThreshold找到的问询索引号
      idxs.append(pos)
      # 移动到问询流中的下一个问询
      pos = pos + 1

  return idxs
```

results:

- input:
- epsilon = 1
- sparse(queries, adult, 3, 0, epsilon)
- output:[18, 21, 22]

7.4 应用：范围问询

范围问询（Range Query）要问的是:”数据集中有多少行的值落在范围中?”范围问询是一种计数问询，因此其敏感度为 1；我们不能对一组范围问询使用并行组合性，因为满足相应问询条件的数据行可能会有重叠。

考虑一组针对年龄列的范围问询（即问询形式为”有多少人的年龄在和 (a,b) 之间?”）。我们可以随机生成很多这样的问询:

```
def age_range_query(df, lower, upper):
  df1 = df[df['Age'] > lower]
  return len(df1[df1['Age'] < upper])

def create_age_range_query():
    lower = np.random.randint(30, 50)
    upper = np.random.randint(lower, 70)
    return lambda df: age_range_query(df, lower, upper)


range_queries = [create_age_range_query() for i in range(10)]
results = [q(adult) for q in range_queries]
```

这些范围问询的回复结果可能相差甚远。部分问询范围可能只会匹配上很少的数据行（甚至匹配不上任何数据行），对应的计数值很小。然而，另一部分问询范围可能会匹配上大量的数据行，对应的计数值很大。在多数情况下，我们知道小计数值的差分隐私回复结果会很不准确，得到这些问询结果的实际意义不大。我们想要做的是了解哪些问询的结果是有价值的，并仅为这些有价值的问询结果支付隐私预算。

我们可以使用稀疏向量技术实现这一点。首先，我们确定一个阈值，并得到范围问询流中回复结果超过此阈值的问询索引号。我们认为这些问询都是”有价值的”问询。随后，我们应用拉普拉斯机制得到这些有价值问询的差分隐私回复结果。这样一来，总隐私开销与超过阈值的问询数量成正比，而非与总问询数量成正比。如果我们预计只有少数问询的回复结果会超过阈值，则所需的总隐私开销会小得多。

```
def range_query_svt(queries, df, c, T, epsilon):
  # 首先，执行sparse，得到"有价值的"问询
  sparse_epsilon = epsilon / 2
  indices = sparse(queries, adult, c, T, sparse_epsilon)

  # 所有，为每个"有价值的"问询执行拉普拉斯机制
  laplace_epsilon = epsilon / (2*c)
  results = [laplace_mech(queries[i](df), 1, laplace_epsilon) for i in indices]
  return results
```

此算法实现中，我们使用一半的隐私预算来确定高于阈值 10000 的前个问询，另一半隐私预算则用于获取这些问询的噪声回复结果。如果高于阈值的问询数量远小于总问询数量，使用此方法就可以获得更准确的回复结果。

第八章 机器学习

本章我们会简要介绍一下如何将差分隐私引入机器学习。我们将重点关注一类特定的监督学习问题：给定一组带标签的训练样本 $\{(x_1, y_1)\ldots\}$，其中 x_i 称为特征向量，y_i 称为标签，我们要训练一个模型 θ。该模型可以预测没有在训练集中出现过的新特征向量所对应的标签。一般来说，每个 x 都是一个描述训练样本特征的实数向量，而 y 是从预先定义好的类型集合中选取的，每个类型一般用一个整数来表示。我们预先要从全部样本中提取出所有可能的类型，构成类型集合。

8.1 噪声梯度下降

在每轮模型更新前在梯度上增加噪声。由于我们直接在梯度上增加噪声，因此该方法通常被称为噪声梯度下降（Noisy Gradient Descent）。

我们的梯度函数是向量值函数，因此我们使用 gaussian_mech_vec（向量高斯机制）在梯度函数的输出值上增加噪声：

```
def noisy_gradient_descent(iterations, epsilon, delta):
    theta = np.zeros(X_train.shape[1])
    sensitivity = '???'

    for i in range(iterations):
        grad = avg_grad(theta, X_train, y_train)
        noisy_grad = gaussian_mech_vec(grad, sensitivity, epsilon, delta)
        theta = theta - noisy_grad

    return theta
```

这里就差一块拼图了：梯度函数的敏感度是多少？这是使算法满足差分隐私的关键所在。

这里我们主要面临两个挑战。第一，梯度是均值问询的结果，即梯度是每个样本梯度的均值。我们之前已经提到，最好将均值问询拆分为一个求和问询和一个计数问询。做到这一点并不难，我们可以不直接计算梯度均值，而是计算每个样本梯度噪声和，再除以噪声计数值。第二，我们需要限制每个样本梯度的敏感度。有两种基础方法可以做到这一点。我们可以（如之前讲解的其他问询那样）分析梯度函数，确定其在最差情况下的全局敏感度。我们也可以（如”采样-聚合”框架那样）裁剪梯度函数的输出值，从而强制限定敏感度上界。

8.2 梯度裁剪

回想一下，在实现”采样-聚合”框架时，我们裁剪未知敏感度函数 $f(x)$ 的输出，强制限定 f 的敏感度上界。f 的敏感度为：$|f(x) - f(x')|$

使用参数 b 裁剪后，上述表达式变为：$|clip(f(x), b) - clip(f(x'), b)|$

最差情况下，$clip(f(x), b) = b$，且 $clip(f(x'), b) = 0$，因此裁剪结果的敏感度为 b（即敏感度等于裁剪参数）。

我们可以使用相同的技巧来限定梯度函数的 l_2 敏感度。我们需要定义一个用来”裁剪”向量的函数，使输出向量的 l_2 范数落在期望的范围内。我们可以通过缩放向量来做到这一点：如果把向量中每个位置的元素都除以向量的 l_2 范数，则所得向量的 l_2 范数为 1。如果想要使用裁剪参数 b，我们可以在缩放后的向量上乘以 b，将其放大回 l_2 范数等于 b 的向量。我们还希望不对 l_2 范数已经小于 b 的向量进行任何修改。因此，如果向量的 l_2 范数已经小于 b，我们直接返回此向量即可。我们可以使用 np.linalg.norm 函数，并以参数 ord=2 作为输入，以计算向量的范数。

```
def L2_clip(v, b):
  norm = np.linalg.norm(v, ord=2)

  if norm > b:
     return b * (v / norm)
  else:
     return v
```

现在，我们可以继续计算裁剪梯度之和，并根据我们通过裁剪技术得到的 l_2 敏感度上界 b 来增加噪声。

```
def gradient_sum(theta, X, y, b):
  gradients = [L2_clip(gradient(theta, x_i, y_i), b) for x_i, y_i in zip(X,y)]
  # 求和问询
  # （经过裁剪后的）L2敏感度为b
  return np.sum(gradients, axis=0)
```

我们现在就快要完成噪声梯度下降算法的设计和实现了。为了计算平均噪声梯度，我们需要：

1. 基于敏感度 b，在梯度和上增加噪声
2. 计算训练样本数量的噪声计数值（敏感度为 1）
3. 用 (1) 的噪声梯度值和除以 (2) 的噪声计数值

```
def noisy_gradient_descent(iterations, epsilon, delta):
  theta = np.zeros(X_train.shape[1])
  sensitivity = 5.0

  noisy_count = laplace_mech(X_train.shape[0], 1, epsilon)

  for i in range(iterations):
     grad_sum      = gradient_sum(theta, X_train, y_train, sensitivity)
     noisy_grad_sum = gaussian_mech_vec(grad_sum, sensitivity, epsilon, delta)
     noisy_avg_grad = noisy_grad_sum / noisy_count
     theta         = theta - noisy_avg_grad

  return theta
```

前面所述方法的普适性很高，因为此方法不需要假设梯度函数满足什么特定的要求。但是，我们有时的确对梯度函数有所了解。特别地，一大类常用的梯度函数（比如对率损失梯度）是利普希茨连续（Lipschitz Continuous）的。这意味着这些梯度函数的全局敏感度是有界的。用数学语言描述，我们可以证明：

$$if \|x_i\|_2 \leq b, then \|\nabla(\theta, x_i, y_i)\|_2 \leq L \tag{8.1}$$

这一结论允许我们通过裁剪训练样本（即梯度函数的输入）来获得梯度函数的 l_2 敏感度上界。这样，我们就不再需要裁剪梯度函数的输出了。

用裁剪训练样本代替裁剪梯度会带来两个优点。第一，与预估训练阶段的梯度尺度相比，预估训练样本的尺度（进而选择一个好的裁剪参数）通常要容易得多。第二，裁剪训练样本的计算开销更低：我们只需要对训练样本裁剪一次，训练模型时就可以重复使用裁剪后的训练数据了。但如果选择裁剪梯度，我们就需要裁剪训练过程中计算得到的每一个梯度。此外，为了实现梯度裁剪，我们不得不依次计算出每个训练样本的梯度。但如果选择裁剪训练样本，我们就可以一次计算得到所有训练样本的梯度，从而提高训练效率（这是机器学习中的常用技巧，这里我们不再展开讨论）。

然而，需要注意的是，还有很多常用损失函数的全局敏感度是无界的，尤其是深度学习中神经网络里用到的损失函数更是如此。对于这些损失函数，我们只能使用梯度裁剪法。

我们只需对算法进行简单的修改，就可以把裁剪梯度替换为裁剪训练样本。在开始训练之前，我们需要先使用 L2_clip（L2 裁剪）函数来裁剪训练样本。随后，我们只需要直接把裁剪梯度的代码移除即可。

```
def gradient_sum(theta, X, y, b):
  gradients = [gradient(theta, x_i, y_i) for x_i, y_i in zip(X,y)]
  # 求和问询
  # （经过裁剪后的）L2敏感度为b
  return np.sum(gradients, axis=0)

def noisy_gradient_descent(iterations, epsilon, delta):
  theta = np.zeros(X_train.shape[1])
  sensitivity = 5.0

  noisy_count = laplace_mech(X_train.shape[0], 1, epsilon)
  clipped_X = [L2_clip(x_i, sensitivity) for x_i in X_train]

  for i in range(iterations):
      grad_sum      = gradient_sum(theta, clipped_X, y_train, sensitivity)
      noisy_grad_sum = gaussian_mech_vec(grad_sum, sensitivity, epsilon, delta)
      noisy_avg_grad = noisy_grad_sum / noisy_count
      theta         = theta - noisy_avg_grad

  return theta
theta = noisy_gradient_descent(10, 0.1, 1e-5)
```

此算法仍然有许多改进空间，如

- 将总隐私消耗量限定为 ϵ，在算法内部计算每轮迭代的隐私消耗量 ϵ_i
- 利用高级组合性、Renyi 差分隐私或 ZcDP，从而获得更好的总隐私消耗量
- 小批次训练：每轮迭代中，不使用整个训练数据，而是使用一小块训练数据来计算梯度（这样可以减少梯度计算过程中的计算开销）
- 同时使用小批次训练和并行组合性
- 同时使用小批次训练和小批次随机采样
- 调整学习率等其他超参数