浙江大学

本科实验报告

课程名称:	数据要素市场
实验名称:	大作业：数据动态定价算法实现
姓　　名:	任朱明　李文耀　江舜尧
学　　号:	
指导教师:	刘金飞
报告日期:	2025/07/22

Contents

Introduction

1 问题背景

如果买家想要买数据训练人工智能，那么用买到的数据训练的模型准确率，显然就表明了数据对于买家的价值，这个价值范围在 $[0,1]$ 之间。根据常理，买的数据越多，训练出来的模型越好，所以数据数量到数据的价值应该是个单调递增的函数。

显然，不同来买数据的人训练的模型不同，他们不同数据量的价值也是不同的。

作为卖家，我们拥有多个等价的数据，现在我们希望可以找到一种给不同数量的数据定价的策略，从而实现收益的最大化。

2 问题建模

首先，我们将完成建模工作，将本问题中需要解决的问题做一个形式化的定义。

2.1 买卖双方都知道的信息（公共信息）

卖家有 N 个完全等价的数据点，因而买卖双方对于数据点的定价和估价只与数据点数量有关。

买家有 m 种类型，第 i 种类型的买家买 n 个完全等价的数据点给买家带来的收益/买家的事前价值/买家的估值曲线为函数 $v_i(n)$，满足：

- 单调递增：没有道理说买了更多数据训练的更差了。
- $v_i(0)=0$：不买数据买家显然没有收益。
- 定义域：$\{0,1,2,...,N\}$（文中表示为 $[N]$）。
- 值域：$[0,1]$。

卖家知道每种买家的收益曲线，但是不知道每种买家的实际比例，或者买家根本没有分布。

2.2 交易过程

一共有 T 轮交易，在第 t 次交易中，卖家会选择一个定价函数 $p_t(n)$，表示第 t 轮交易卖出 n 个数据点的价格。这轮交易中买方的类型为 i_t。买方会选择一个购买商品的数量，

使得此时自己获得最大的利润，即买方的估值函数减去卖方估值函数 $v_{i_t}(n) - p_t(n)$ 在这个点最大，并且这个值为正数。

如果无法获得正的利润，即买哪个商品期望值都低于付出的钱，那么买家不会告诉卖家自己的类型，也不会购买，而当且仅当买家能获得正数的利润，买家会购买，并在完成购买后告诉卖家自己的类型。

定价函数定义域为 $\{0, 1, 2, ..., N\}$ 文中表示为 $[N]$。值域为 $[0, 1]$。

2.3 一些常用符号的定义

- $[N]$：卖家定价函数和买家估值函数的定义域
- m-step ：一个函数 f 对于 $[N]$ 到 $[0, 1]$ 的单调不减函数，满足至多有 m 个点 $n \in \{0, 1, 2, ..., N-1\}, f(n+1) - f(n) > 0$。
- $\mathcal{P}$：所有可能的定价函数的集合。
- $n_{i,p}$：假设某场交易中，卖方的定价函数/曲线是 p，买方的类型为 i，那么买方会选择购买商品的数量为 $n_{i,p}$。
- q：令买家类型的分布为 q，其中第 i 种买家的概率为 q_i。
- $\mathrm{rev}(p)$：卖家卖一件商品的期望收入，即按照买家类型算期望，满足

$$\mathrm{rev}(p) = \sum_{i=1}^{m} q_i p_i\big(n_{p,i}\big)$$

- p^{OPT}：最优定价曲线
- $\mathrm{rev}\big(p^{\mathrm{OPT}}\big)$：按最优定价卖一件商品期望收益

2.4 买家估值函数的特殊性质

本题中买家所有的估值函数都是：

- 单调不减
- 定义域为 $\{0, 1, 2, ..., N\}$
- 值域为 $[0, 1]$
- $v_i(0) = 0$。

但是在实际的应用中，买家的估值函数常常满足以下两条性质，使得如果我们假设以下两条性质成立，可以用更加高效的算法实现相近的性能。

接下来我们来介绍这两个性质。

2.4.1 假设 1：买家的估值函数是平滑的

对于买家的估值函数，相近的数据量效用不会差太多：

$$\exists L > 0, \forall n, n' \in \{0, 1, 2, ..., N\}, v_i(n + n') - v_i(n) \leq \frac{L}{N} n'.$$

上式中 L 是某个正常数，我们将满足上述条件称为 L 阶 Lipschitz 平滑条件（L-Lipschitz-like smoothness condition）。

2.4.2 假设 2：数据的效用是边际效用递减的

买家买的数据越多，数据越不值钱：

$$\exists J > 0, \forall i \in \{0, 1, 2, ..., m\}, \forall n \in \{0, 1, 2, ..., N\}, v_i(n+1) - v_i(n) \leq \frac{J}{n}$$

上式中 J 是某个正常数，称为边际收益递减常数。

2.5 任务场景

2.5.1 随机环境

这种任务中，买家的概率分布 q 是固定的，第 t 轮来买数据的买家类型是按照这个分布随机抽取的 i_t。

这种场景下，遗憾值为最优定价曲线总的期望收益，减去实际上这 T 轮的卖数据的历史收入。

$$R_T = T \cdot \mathrm{OPT} - \sum_{t=1}^{T} p_t\left(n_{i_t,p_t}\right)$$

2.5.2 对抗场景

这种任务中，第 t 轮来买数据的买家类型是一个预先选好的随机序列。

这种场景下，遗憾值为根据这个对手的策略选择的最优策略的期望收益减去实际的收益。

$$R_T = \max_{p \in \mathcal{P}} \sum_{t=1}^{T} P\left(n_{i_t,p}\right) - \sum_{t=1}^{T} p_t\left(n_{i_t,p_t}\right)$$

3 前人成果

前人成果大多集中于卖家向买家出售单一商品。

前人已经对于买家至多只买一件商品时，已知买家的概率分布，即有多大概率遇上对产品有一种估值的买家，如何分配自己的多种商品的价格。本文则着重研究相同的商品但是买卖多个的情况，而且买家的类型分布不是已知的。

同时，前人的研究大多集中于离线的或者只买卖一次的场景，但是本文研究了在线、买卖多次的算法。

前人研究了一些从无数的定价函数中选择少量定价函数的方法，本文产出了新的研究方法。

4 总结

4.1 本文贡献

4.1.1 多个选择卖家定价函数的方法

因为显然卖家的定价曲线/函数有无穷多个，因为从 N 个自然数映射到 [0,1] 的曲线是无限多的，这太多了，因此本文设计出了一些选择方法，能选择出一些数量适当且收益较为高的函数。

针对平滑性、边际收益递减等特性，设计了三种选择方案

4.1.2 近似最优解的计算

基于 UCB 和 FTPL 等经典方法，结合上述选择方案，提出适用于随机环境和对抗环境的在线学习算法，通过平衡即时收益与信息获取，在多轮市场中逐步逼近最优定价

算法设计确保当买方类型数量固定时，计算复杂度与数据点数量呈多项式关系，因此在类型数量有限的大型数据集场景中具有实际可行性。

4.2 未来展望

文中提出的一个关键未来研究方向是，即卖方不知道买方估值曲线 v_i 的假设。当前模型假设卖方知晓各类型买方的估值曲线，而实际场景中，买方估值往往是未知的，如何在缺乏先验估值信息的情况下实现最优定价，将是进一步拓展研究的重要方向，这也能让模型更贴近真实市场环境。

Paper Review

文章除了回顾过往研究以及进行了一些总结与展望，其主要内容集中于如何求出商家的最优定价策略，即让商家在不断的买卖中算出一个定价，使得自己挣到最多的钱。

文章的总体思路是先在无限个定价策略中取出少量合适且够好的定价策略，然后使用两种不同的方法通过迭代法最终获得选出的定价策略中最好的的那个。

对于不同的假设，即是否认为定价策略是平滑的，是不是有边际效用递减，文章使用的不同的定价策略挑选方法。

文章正文和附录包含了对于这些算法“为什么好”的证明。

1 选择哪些卖家的定价函数

因为显然卖家的定价曲线/函数有无穷多个，因为从 N 个自然数映射到 [0,1] 的曲线是无限多的，我们需要：

1. 用“算法”从中选择有限个简单曲线/函数，每个简单函数/曲线只有至多 m 个 $f(n+1)-f(n)>0$ 的点（即 m-step 的，这就是后文用到的 m-step 的定义）。
2. 证明这些定价函数够用，即对估值造成的损失不大，且总数不能太多。
3. 接下来会分别先证明一个重要引理，然后按照三个附加假设逐个分析在越来越严格的定价函数限制下我们如何选择定价函数，并证明这些函数足够好。

1.1 准备工作

m-step ：一个函数 f 对于 $[N]$ 到 $[0,1]$ 的单调不减函数，满足至多有 m 个点 $n\in\{0,1,2,...,N-1\}, f(n+1)-f(n)>0$。

Lemma 1

可以用简单的 m-step 函数代替所有的卖家估值函数。具体地，如果有 m 种买家，对于任何一个卖家的非递减定价函数，都有 m-step 函数收益不低于这个定价函数。

这个引理的内容是如果有 m 种买家，对于任何一个卖家的非递减定价函数，都有 m-step 函数收益不低于这个定价函数。

这个引理的内容的证明方法在 A.1 中给出。主要的思路是构造出一个函数 $\bar{p}$，使得 $\bar{p}$ 是 m-step 的，并且 $\bar{p}$ 的收益不低于原函数。思路是把没人买的数据量通通涨价，直到函数被简化为阶梯函数。

不妨让没有买数据的人买的数据量 $n_{i,p}=0$，买了数据的人买了 $n_{i,p}$ 个数据。我们将所有的 $n_{i,p}$ 排序，得到一个递增数列 $\left\{n_{(i)}\right\}$。接着构造函数 $\overline{p}$，若 $n_{(i)}<n\leq n_{(i+1)}$，则 $\overline{p}=p\left(n_{(i+1)}\right)$，最后大于 $n_{(m)}$ 的数据量的价格为 $p(N)$。感性理解就是有人买的数据量不涨价，没人买的数据量都涨价到一个有人买的数据量的价格。

这样，所有按照 $\overline{p}$ 计算的数据定价除了有顾客买的数据量价格不变，价格都高于原函数。显然对于每一个买家，买没有涨价的数据收益显然不如买原来买的，不然原来就不会买这个了，对于涨价的数据量，数据没涨价都不会买涨价了更不会买了，所以顾客不会改变购买选择。由于顾客购买数据量的定价是不变的，所以我们卖出这些数据收入就没有变化。

而由于顾客只有 m 种类型，所以 $\overline{p}$ 至多会在 m 个点上有跳变，这是由于我们的构造方法决定的，跳变的点有且仅有顾客买数据的这至多 m 个点。所以 $\overline{p}$ 至多会在 m 个点上有跳变，是 m-step 函数。

m-step 函数不只有我们构造出的 $\overline{p}$，还可以有其他的 m-step 函数。所以必然有一收益更高的 m-step 函数能代替所有的卖家估值函数。

1.2 算法及其正确性证明

我们先给出好的估价函数选择算法的定义：

Definition

一个估价函数选择算法是好的，如果其选出的函数收益够高，数量也够小。具体地，对于任意买家类型分布，该算法可以选出一个估价函数集合 $\overline{\mathcal{P}}$，满足

1. 存在 $p\in\overline{\mathcal{P}}$，使得其收益 $\mathrm{rev}(p)\geq \mathrm{OPT}-O(\varepsilon)$，即与最优收益的差距足够小。
2. 集合大小 $|\overline{\mathcal{P}}|$ 满足一定的有界性要求，即选出的函数足够少。

下面我们将逐步加强定价函数的条件，并给出在这些条件下好的估价函数选择算法。

- 条件：估值函数满足单调递增且买家种类 m 有限
- 输入：误差范围 ε，买家种类 m
- 过程：

  首先用等比数列 Z_i 对于卖家估值函数的值域作切分。

  $$Z_i=\left\{\varepsilon(1+\varepsilon)^i,\forall i=0,1,2,...,\left\lceil\log_{1+\varepsilon}\frac{1}{\varepsilon}\right\rceil\right\}$$

  接着对每一个部分作插值进一步细化值域

  $$W_i=\left\{Z_{i-1}+Z_{i-1}\cdot\frac{\varepsilon k}{m}\mid k=1,2,3,...,\lceil(2+\varepsilon)m\rceil\right\}$$

  $$W=\bigcup_{\{i=1\}}^{\lceil\log_{1+\varepsilon}\frac{1}{\varepsilon}\rceil}W_i$$

  然后输出 $\overline{\mathcal{P}}$，表示所有从 $\{0,1,2,...,N\}$ 到 W 的 m-step 映射。

Theorem

算法 1 是好的估价函数选择算法，不仅损失够小，$\exists p \in \overline{\mathcal{P}}, \mathrm{rev}(p) \geq \mathrm{OPT} - \mathcal{O}(\varepsilon)$，而且选出的函数数量足够少，$\overline{\mathcal{P}} \in \tilde{\mathcal{O}}\left(\left(\frac{N}{\varepsilon}\right)^m\right)$。

下面我们给出该算法是好的证明：

Proof

我们先提出并证明三个引理：

1. **Lemma A.1**

   $\forall \varepsilon$，存在定价函数 $\tilde{p} : [N] \to [\varepsilon, 1]$，使得 $\mathrm{rev}(\tilde{p}) \geq \mathrm{OPT} - \varepsilon$。

   设最优定价函数 OPT 为 p^*，将 p^* 中价格低于 ε 的点全部涨价到 ε，得到函数 $\tilde{p}$。即 $p^*(n) = \max(p^*(n), \varepsilon)$。

   如果买家购买的数据量价格不变，那么买家带来的收益不变，但是如果买家购买的数据量价格涨价，那么即使所有买家买的数据都涨价导致买家都不买了，那么损失的期望也是 $\varepsilon \times 1 = \varepsilon$，所以卖一次的期望收益最多减少 ε，即引理成立。

2. **Lemma A.2**

   在使用算法 1 从无穷个定价函数中选出的 $\overline{\mathcal{P}}$ 中，有一函数离刚刚构造出的 $\tilde{p}$ 足够近。即 $\exists p \in \overline{\mathcal{P}}$，使得 $\mathrm{rev}(p) \geq \frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon}$。

   由于引理 1 ，我们可以找到一个函数 $\bar{p}$ 使得 $\bar{p}$ 是 m-step 的，并且 $\bar{p}$ 的收益不低于原函数。我们将 $\bar{p}$ 中的所有取值都下降到算法 1 中构造的 W 中的最靠近的值，构造出 p，这个 p 显然在 $\overline{\mathcal{P}}$ 中，因为本身 $\overline{\mathcal{P}}$ 中就包含了所有值域为 W 的 m-step 函数，下面将证明我们构造的 p 与 $\tilde{p}$ 相差不大。

   如果不需要下降，那么自然收益不变。如果需要下降，则由于 W_i 中间的插值是 $Z_{i-1} \cdot \frac{\varepsilon}{m}$，且 Z_i 是以 $(1+\varepsilon)$，为公比的等比数列，因而 $\frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon}$ 到 $\mathrm{rev}(\tilde{p})$ 之间显然会有一个 Z_i，那也必定有 W。

   所以如果买家不改变购买数量，收益为降价之后的价格，由于这个值在 $\frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon}$ 到 $\mathrm{rev}(\tilde{p})$ 之间，所以期望价格大于 $\frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon}$。

   那么买家会不会买更少的数据导致更小的卖家收益呢？答案是不会。因为根据 Z 的构造方法，Z_i 是等比数列，这导致 Z_i 越大越稀疏，而 W_i 是在 Z_i 的基础上插值的，因而 W_i 越大也会越来越稀疏。而数据卖的越多价格越贵，因此价格越高降价越多，自然买家只会不改变购买数量或者买更多的数量。这样要么收益增加，要么收益大于 $\frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon}$。

3. **Lemma A.3**

   从无穷个卖家定价函数中用算法 1 选出的数量够少。只要满足 $n > m$ 的前提，就有

   $$|\overline{\mathcal{P}}| \leq \left(\frac{eN}{m}\right)^m \cdot \left(e\lceil 2+\varepsilon \rceil \left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rceil\right)^m$$

利用排列组合，先列求和公式，假设函数有 i 个跳跃点，每个跳跃点是在 $|N-1|$ 的组合数。跳跃点的取值就是 $|W|$ 个已知值域的组合。

$$|\overline{\mathcal{P}}| = \sum_{i=1}^{m} \binom{N-1}{i} \binom{|W|}{i}$$

因为以下式子展开之后显然包含了以上式子加上某个正数，所以显然有

$$\begin{aligned} |\overline{\mathcal{P}}| &\leq \left(\sum_{i=1}^{m} \binom{N-1}{i} \right) \left(\sum_{i=1}^{m} \binom{|W|}{i} \right) \\ &\leq \left(\sum_{i=0}^{m} \binom{N-1}{i} \right) \left(\sum_{i=0}^{m} \binom{|W|}{i} \right) \\ &\leq \left(\frac{e(N-1)}{m} \right)^m \cdot \left(\frac{e\,|W|}{m} \right)^m \end{aligned}$$

其中最后一个不等号使用了组合数不等式放缩。由 W 的定义，$\left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rceil$ 个 Z_i 每个切分为 $\lceil 2+\varepsilon \rceil$ 个点，所以 $|W|$ 至多为 $\lceil 2+\varepsilon \rceil \left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rceil$，故有

$$|\overline{\mathcal{P}}| \leq \left(\frac{e(N-1)}{m} \right)^m \cdot \left(e\lceil 2+\varepsilon \rceil \left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rceil \right)^m.$$

由引理 A.1 和引理 A.2，我们可以得到：

$$\mathrm{rev}(p') \geq \frac{\mathrm{rev}(\tilde{p})}{1+\varepsilon} \geq \frac{\mathrm{OPT}-\varepsilon}{1+\varepsilon}$$

由于最佳收益 $\mathrm{OPT} \in [0,1]$，故有 $\varepsilon \geq \varepsilon\ \mathrm{OPT}$，即 $\mathrm{OPT}-\varepsilon \geq (\varepsilon+1)\mathrm{OPT}-2\varepsilon$，因而

$$\mathrm{rev}(p') \geq \frac{\mathrm{OPT}-\varepsilon}{1+\varepsilon} \geq \mathrm{OPT} - \frac{2\varepsilon}{1+\varepsilon} = \mathrm{OPT} - O(\varepsilon),$$

即满足好的算法的第一个条件。

在引理 A.3 的基础上，由于 $\log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \ll \frac{1}{\varepsilon}$，所以

$$\left(\frac{e(N-1)}{m} \right)^m \leq N^m,$$

$$\left(e\lceil 2+\varepsilon \rceil \left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right) \right\rceil \right)^m \leq \varepsilon^{-m}$$

所以 $|\overline{\mathcal{P}}| \in \tilde{O}\left(\left(\frac{N}{\varepsilon}\right)^m\right)$，即满足好的算法的第二个条件。

综上，算法 1 是好的估价函数选择算法。

1.2.1 算法 5

- 条件：估值函数单调递增，满足平滑性，且买家种类 m 有限

- 输入：误差范围 ε，买家种类 m，平滑常数 L，满足

$$\forall n, n' \in \{0, 1, 2, ..., N\}, v_i(n + n') - v_i(n) \leq \frac{L}{N} n'$$

- 过程：

  首先用等比数列 Z_i 对于卖家估值函数的值域作切分：

$$Z_i = \left\{ \varepsilon(1+\varepsilon)^i, \forall i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil \right\}$$

  接着对每一个部分作插值进一步细化值域：

$$W_i = \left\{ Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \right\}$$

$$W = \bigcup_{\{i=1\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$$

$W_i = \left\{ Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \mid k = 1, 2, 3, ..., \lceil (2+\varepsilon)m \rceil \right\}$

$W = \cup_{\{i=1\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$

以上步骤和算法 1 完全相同。

然后把 $[0, N]$ 划分为均匀网络，m-step 函数的跳跃点仅取这个划分中的点。

离散化间隔 $\delta = \lfloor \frac{\varepsilon N}{mL} \rfloor$ 表示划分的长度。

定义跳跃点可以取的点为 $N_s = \left\{ \delta k : k \in \left\{ 0, 1, 2, ..., \lceil \frac{N}{\delta} \rceil \right\} \right\}$。

最后输出 $\overline{\mathcal{P}}$，表示所有从 N_s 到 W 的 m-step 映射。

Theorem 3.2

算法 5 是好的估价函数选择算法，不仅损失够小，$\exists p \in \overline{\mathcal{P}}, \text{rev}(p) \geq \text{OPT} - \mathcal{O}(\varepsilon)$，而且选出的函数数量足够少，$|\overline{\mathcal{P}}| \in \tilde{\mathcal{O}}\left(\left(\frac{L}{\varepsilon^2}\right)^m\right)$。

证明过程如下：

Proof

1. 构造 m-step 函数

   由引理 3.1：存在一个至多 m 个跳跃点的函数 p^*，使得 $\text{OPT} \leq \text{rev}(p^*)$。

   如果 p^* 中相邻的阶梯值之差小于 $\frac{\varepsilon}{m}$ 那么消去这个阶梯，并且连着后面的所有函数值一起调整。

   详细的来说，对于每个函数值，函数 p^* 中这个点左边有多少个跳跃点的阶梯值之差小于 $\frac{\varepsilon}{m}$，那么它的值就应该减少这些跳跃点差的值之和。这样就构造出了函数 p'。

   这一步相当于降价，并且越贵的数据量降价越多，那么原本卖数据的客户仍然会选择买至少不少于原先的数据量。

每一名客户少付的钱最多为函数下降的值，最多下降 m 次，每次最多 $\frac{\varepsilon}{m}$ ，所以总的来说 p' 对收益的期望即 OPT 带来的损失不超过 ε。

2. 构造 $\overline{\mathcal{P}}$ 中的函数 p

   p' 可以用它的 k 个跳跃点来表示，$p' = \{p'(n_i)\}$，其中 n_i 是 p' 的跳跃点。

   我们将 n_i 修改为 N_s（这是我们刚刚求出的 $\overline{\mathcal{P}}$ 中可以跳变的点集）中比 n_i 小的最大的点，将 $p'(n_i)$ 调整为 $p'(n_i) - i\frac{\varepsilon}{m}$ 附近的某个 W 中的值。用新的这 k 个跳变点，我们得到了 m-step 函数 p。

   因为 W 中两个点之间距离为 $\frac{\varepsilon}{m}$ ，所以 $p'(n_i)$ 调整为 $p'(n_i) - i\frac{\varepsilon}{m}$ 附近的某个 W 中的值，这就保证了 p 属于 $\overline{\mathcal{P}}$。并且显然也是买的数据越多降价越厉害。

3. 证明 p 与 OPT 相差不大，选出的函数收益够高。

   买家会不会因为我们向左调整跳变点，导致这一段距离数据涨价，导致买家购买的数据量减少或者不买呢?

   假设 n_i 是 p' 下第 i 个跳跃点 n_i' 在数据网格 N_S 上的取整结果，由于平滑性，买家在这两个点上差距不多，$v(n_i) \geq v(n_i') - \delta\frac{L}{N}$，因此这一段相当于先进行了一个至多不超过 $\frac{L}{N}$ 乘区间长度即 $\frac{L}{N} \cdot \frac{\varepsilon N}{mL} = \frac{\varepsilon}{m}$ 的涨价，又多了一次（因为相当于从左边一个区间到了右边一个区间）大小为 $\frac{\varepsilon}{m}$ 的降价，所以总的来说降价还是多了了，并且降价的幅度在自己的左侧区间和右侧区间之间。总的来说还是符合数据越多降价越多的性质，因此买家显然不会降低自己的数据购买量，因为本来更少的数据就不如原先的选择有吸引力，而且打折力度也更小。

   假如买家不会因为价格改变而减少数据购买，从 p' 到 p 的损失为 $i\frac{\varepsilon}{m} \leq m\frac{\varepsilon}{m}$，所以 p 与 p' 相差至多为 ε。

   假如买家增加数据购买量，显然亏的钱不会多于买家购买量不变。

   综上所述，p 与 p' 相差至多为 ε。

   由于 $p' \geq \mathrm{OPT} - \varepsilon$，所以 $p \geq \mathrm{OPT} - 2\varepsilon = \mathrm{OPT} - \mathcal{O}(\varepsilon)$。

4. 证明算法 5 选出的函数数量够少

   由于本构造方法数量仅仅相当于缩小了自变量取值，因而直接沿用上一个结果

   $|\overline{\mathcal{P}}| \leq \left(\frac{e(N-1)}{m}\right)^m \cdot \left(e\lceil 2+\varepsilon\rceil\left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right\rceil\right)^m$

   将 N 替换为 $|N_s| \leq \frac{N}{\delta} \leq \frac{mL}{\varepsilon}$ 即可。

   $|\overline{\mathcal{P}}| \leq \left(\frac{e(|N_s|-1)}{m}\right)^m \cdot \left(e\lceil 2+\varepsilon\rceil\left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right\rceil\right)^m$

$W_i = \left\{Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \mid k = 1, 2, 3, ..., \lceil(2+\varepsilon)m\rceil\right\}$ $\left(\frac{e(|N_s|-1)}{m}\right)^m$ 去掉所有定值就是放缩为 $\left(\frac{L}{\varepsilon}\right)^m$，$\left(e\lceil 2+\varepsilon\rceil\left\lceil \log_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right\rceil\right)^m$ 放缩为 ε^{-m}，所以 $|\overline{\mathcal{P}}| \in \tilde{O}\left(\left(\frac{L}{\varepsilon^2}\right)^m\right)$。

1.2.2 算法 2

- 条件：估值函数单调递增，满足边际效用递减的限制，且买家种类 m 有限

- 输入：边际收益递减常数 J，误差范围 ε，买家种类 m
- 过程：

  首先用等比数列 Z_i 对于卖家估值函数的值域作切分。

$$Z_i = \left\{ \varepsilon(1+\varepsilon)^i, \forall i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil \right\}$$

  接着对每一个部分作插值进一步细化值域

$$W_i = \left\{ Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \right\}$$

  以上步骤和算法 1 完全相同。

$$W = \bigcup_{\{i=2\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$$

  对于定价函数的定义域 $[N]$，我们先用一个等比数列分割：

$$Y_i = \left\lfloor \frac{2Jm}{\varepsilon^2}(1+\varepsilon^2)^i \right\rfloor \sim i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon^2} \frac{N\varepsilon^2}{2Jm} \right\rceil$$

  对于每个小范围 $[Y_i, Y_{i+1})$，用步长 $Y_i \cdot \frac{\varepsilon^2}{2Jm}$ 进行插值

$$Q_i = \left\{ Y_i + Y_i \cdot \frac{\varepsilon^2 k}{2Jm}, k = 0, 1, 2, 3, ..., \lfloor 2Jm \rfloor \right\}$$

$$Q = \bigcup_{\{i=1\}}^{\left\lceil \log_{1+\varepsilon^2} \frac{N\varepsilon^2}{2Jm} \right\rceil} Q_i$$

  定义可以取跳跃点的点为 $N_D = \left\{1, 2, ..., \left\lfloor \frac{2Jm}{\varepsilon^2} \right\rfloor \right\} \bigcup Q$，挑选出的卖家定价函数集合为 $\overline{\mathcal{P}}$ 表示所有只能在 N_D 跳变，定义域在 $[N]$ 且取值只能在 W 的单调不减 m-step 函数。

Theorem 3.3

算法 2 是好的估价函数选择算法，不仅损失够小，$\exists p \in \overline{\mathcal{P}}, \text{rev}(p) \geq \text{OPT} - \mathcal{O}(\varepsilon)$，而且选出的函数数量足够少，$|\overline{\mathcal{P}}| \leq O\left(\left(\frac{J}{\varepsilon^2}\right)^m \log^m\left(\frac{N\varepsilon^2}{Jm}\right) \cdot \left(\log^m_{1+\varepsilon}\left(\frac{1}{\varepsilon}\right)\right)\right) \in \tilde{O}\left(\left(\frac{J}{\varepsilon^3}\right)^m\right)$。

证明过程如下：

Proof

首先我们需要构造出一个 $p \in \overline{\mathcal{P}}$ 再证明定理的成立。

根据 Lemma 3.1，存在一个卖家的最优定价 p^*，它是一个 k-step 函数 $(k \leq m)$，可用每个跳跃点的坐标表示为 $(n_1^*, p_1^*), (n_2^*, p_2^*), ..., (n_k^*, p_k^*)$，其中 n_i^* 是跳跃点 ，p_i^*是对应区间的价格。

$p^*(n_i)$ 调整为 $p^*(n_i) - iZ_{n_i^*-1}\frac{\varepsilon}{m}$ 附近的某个 W 中的值，这和上一种方法的证明是一样的，越贵降价越多，而且可以确保函数仍然是单调不减的。

接下来我们将自变量 n_i^* 调整为 N_D 中最接近自己左侧的值。

自变量 n_i^* 调整为 N_D 中自己左侧最接近的值，变化不会超过 $Y_i\frac{\varepsilon^2}{2Jm}$，由于边际效用递减，这个点左移造成的涨价为 $\frac{J\varepsilon^2}{2Jm} = \frac{\varepsilon^2}{2}m$，但是它多降价的 $Z_{n_i^*-1}\frac{\varepsilon}{m} \geq \varepsilon \cdot \frac{\varepsilon}{m}$，所以它的降价仍然多于原来在其左侧的点的降价。

所以，仍然是买的越多降价越多，顾客不会减少购买量，所以损失只需要考虑降价的损失即可。

降价的最高期望损失为 $\frac{\varepsilon^2}{m} * m = \mathcal{O}(\varepsilon)$。

下面证明 $|\overline{\mathcal{P}}|$ 足够小，选出的函数足够少：

数据量集合 N_D 的大小：小数据量部分约 $\frac{2Jm}{\varepsilon^2}$ 个点，

大数据量部分按指数间隔划分，点数约为 $\mathcal{O}\Big(Jm\log_{1+\varepsilon^2}\frac{N\varepsilon^2}{Jm}\Big)$，

总大小为 $O\big(\frac{Jm}{\varepsilon^2} + Jm\log N\big)$，忽略增长缓慢的对数，那么$|N_D| = \mathcal{O}\big(\frac{2Jm}{\varepsilon^2}\big)$。

按照前两个证明过程一模一样的做法，相当于 $\Big(\frac{e(N-1)}{m}\Big)^m \cdot \big(e\lceil 2+\varepsilon\rceil\lceil\log_{1+\varepsilon}\big(\frac{1}{\varepsilon}\big)\rceil\big)^m$ 中，$\big(e\lceil 2+\varepsilon\rceil\lceil\log_{1+\varepsilon}\big(\frac{1}{\varepsilon}\big)\rceil\big)^m$ 仍然被放缩为 ε^{-m}，$\Big(\frac{e(N-1)}{m}\Big)^m$ 放缩为 N^m，但是这里的 N 变成了 $|N_D| = \mathcal{O}\big(\frac{2Jm}{\varepsilon^2}\big)$，放缩为 $\mathcal{O}\Big(\big(\frac{J}{\varepsilon^2}\big)^m\Big)$，所以最终 $|\overline{P}| \in \tilde{O}\Big(\big(\frac{1}{\varepsilon}\big)^m\Big) \cdot \mathcal{O}\Big(\big(\frac{J}{\varepsilon^2}\big)^m\Big)$。

所以 $|\overline{P}| \in \tilde{O}\Big(\big(\frac{J}{\varepsilon^3}\big)^m\Big)$。

2 买家类型存在一个概率分布时的卖家定价算法

作为卖家方，我们回顾一下我们的挑战。首先我们知道买家有一个概率分布，但是我们不知道买家的概率分布是什么。知道了买家的概率分布我们就可以在之前选出的一系列定价函数 $\overline{\mathcal{P}}$ 中选出一个让我们挣钱最多的。

所以我们的算法任务是，根据之前来的买家的结果，估计出买家的分布，接着选出一个合适的定价函数来挣钱。

作为研究者，我们需要证明这个算法求出的函数是足够好的。

2.1 算法 3：针对有概率分布的买家如何选出挣钱最多的定价

买家类型的概率分布固定，但是卖家不知道这个具体的概率分布，只知道买家的种类数量和每种买家的估价函数，通过每轮选择计算出的“可能最大收益”（即收益的上置信界）最大的定价曲线。

这样既可以按照历史经验最大化本轮的收益，也可以逐渐根据历史经验算出哪个曲线是最好的，逐步逼近最优定价。

同时，算法中也近似的算出来买家种类的分布。

算法需要输入来购买的轮数 T，之前选出来的卖家比较好的定价函数集合 $\overline{\mathcal{P}}$。买方会选择一个购买商品的数量，使得此时自己获得最大的利润，即买方的估值函数减去卖方估值函数

第一轮先免费卖一次数据，显然这个类型为 i_1 的买家会买走所有 N 个数据点。

2 到 T 轮中的第 t 轮分别进行以下步骤:

2.1.1 用 UCB 法计算 $\overline{\mathcal{P}}$ 中每个卖家定价函数 p 的最大期望收益 $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$

计算步骤如下:

1. S_τ 表示“第 τ 轮有意向购买的买家编号”，是一个集合。含义是按照第 τ 轮卖家的历史定价函数 p_τ，会买数据的买家，他们的编号的集合。即如果卖家的函数上存在一点 n_i 使得 $v_i(n_i) \geq p_\tau(n_i)$，即如果本轮参与购买可以获益，那么这个买家的编号就会包含在 S_τ 中。
2. $T_{i,t}$ 表示“前 t 编号为 i 的买家本应该购买的次数”，即按照前 t 轮卖家的历史定价，如果类型为 i 的买家全去买了一遍会在几轮中购买。因为第一轮是白送的，所以买家至少可以在这一次购买中获益，所以如果每次购买都参加那么至少买这一次，即 $T_{i,t} \geq 1$。
3. $\overline{q}_{i,t}$ 表示”根据前 t 轮结果估计的类型 i 的买家的比例”。具体算法是实际上这 t 轮来购买数据的 i 类型买家数量，除以“前 t 轮编号为 i 的买家本应该购买的次数”。
4. $\hat{q}_{i,t}$ 表示“根据前 t 轮结果估计出的买家比例上界”，即 $\overline{q}_{i,t}$ 加上一个长得很像 UCB 经典公式的妙妙公式，该公式和总买卖轮次 T 和“本应该购买的次数“有关。公式形式为 $\hat{q}_{i,t} = \overline{q}_{i,t} + \sqrt{\frac{\log(T)}{T_{i,t}}}$
5. 计算“最大期望收益” $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$，即按照“根据前 t 轮结果估计出的买家比例上界”估计出收益的上界，方法是用$\hat{q}_{i,t}$ 这个“根据前 t 轮结果估计出的买家比例上界”计算收益的期望。 $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p) = \sum_{i=1}^{m}\big(\hat{q}_{i,t} \cdot p\big(n_{i,p}\big)\big)$

2.1.2 选出本轮采用的卖家定价函数 p

选出上一步算出的“最大期望收益” $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$ 最大的那个 p，这就是本轮卖家的定价 p_t

2.1.3 完成买卖

类型为 i_t 的买家会买走 n_{i_t,p_t} 个数据点，并付钱 $p_t\big(n_{i_t,p_t}\big)$。

其中 i_t 是从买家的分布中按照概率随机抽取的。

2.2 定理 4.1：算法的期望遗憾足够小

我们只要将 $\overline{\mathcal{P}}$ 取得误差足够小，这是我们在三个选择算法中均保证过的，因为我们可以根据输入的最大误差来用不同的方法选取函数，那么可以保证用算法 3 造成的遗憾值足够小。

定理的内容是：只要计算 $\overline{\mathcal{P}}$ 用的误差上界 $\varepsilon \in \mathcal{O}\Big(\frac{1}{\sqrt{T}}\Big)$ 那么使用算法 3 造成的遗憾的期望

$$\mathbb{E}[R_T] \in \tilde{O}\big(m\sqrt{T}\big)$$

2.2.1 步骤 1：切分遗憾

将遗憾值 R_T 从最优值-实际值改写为（最优值-可能达到的最优值）+（可能达到的最优值-实际值），即 $T \cdot \mathrm{OPT} - T \cdot \max_{p\in\overline{\mathcal{P}}} \mathrm{rev}(p) + T \cdot \max_{p\in\overline{\mathcal{P}}} \mathrm{rev}(p) - \sum_{t=1}^{T} p_t\big(n_{i_t,p_t}\big)$

$T \cdot \mathrm{OPT} - T \cdot \max_{p\in\overline{\mathcal{P}}} \mathrm{rev}(p)$ 这个部分表示由于我们从所有定价函数中选了一部分，导致可能实际上达不到最优，实际的最优与我们的最优差了多少呢?

我们在选择函数的算法中那一部分已经知道了这个差距范围是 $T \cdot \mathcal{O}(\varepsilon) = T \cdot \mathcal{O}\Big(\frac{1}{\sqrt{T}}\Big) = \mathcal{O}\big(\sqrt{T}\big)$

接下来我们只需要研究在我们选出的 $\overline{\mathcal{P}}$ 集合中，我们逼近可以选择的最优函数的过程造成了什么损失了。

2.2.2 步骤二：引理 C.1：逼近最优选择的过程中不会造成太大误差

这个引理直接研究 $\mathbb{E}[T \cdot \max_{p\in\overline{\mathcal{P}}} \mathrm{rev}(p) - \sum_{t=1}^{T} p_t\big(n_{i_t,p_t}\big) \le \tilde{O}\big(m\sqrt{T}\big)$

3 买家类型被对手提前确定时的卖家定价算法

显然，作为买家方，我们可以针对卖家的定价方法针对性地做局，安排相应类型的买家使得卖家挣不到钱。例如可以先压价在买，或者先抬价再不买。这就需要卖家用一些方法来解决。

我们只要将 $\overline{\mathcal{P}}$ 取得误差足够小，这是我们在三个选择算法中均保证过的，因为我们可以根据输入的最大误差来用不同的方法选取函数，那么可以保证用算法 3 造成的遗憾值足够小。

定理的内容是：只要计算 $\overline{\mathcal{P}}$ 用的误差上界 $\varepsilon \in \mathcal{O}\Big(\frac{1}{\sqrt{T}}\Big)$ 那么使用算法 3 造成的遗憾的期望

$$\mathbb{E}[R_T] \in \tilde{O}\big(m\sqrt{T}\big)$$

3.0.1 步骤 1：切分遗憾

将遗憾值 R_T 从最优值-实际值改写为（最优值-可能达到的最优值）+（可能达到的最优值-实际值），即 $T \cdot \mathrm{OPT} - T \cdot \max_{p \in \overline{\mathcal{P}}} \mathrm{rev}(p) + T \cdot \max_{p \in \overline{\mathcal{P}}} \mathrm{rev}(p) - \sum_{t=1}^{T} p_t\left(n_{i_t,p_t}\right)$

$T \cdot \mathrm{OPT} - T \cdot \max_{p \in \overline{\mathcal{P}}} \mathrm{rev}(p)$ 这个部分表示由于我们从所有定价函数中选了一部分，导致可能实际上达不到最优，实际的最优与我们的最优差了多少呢?

我们在选择函数的算法中那一部分已经知道了这个差距范围是 $T \cdot \mathcal{O}(\varepsilon) = T \cdot \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) = \mathcal{O}\left(\sqrt{T}\right)$

接下来我们只需要研究在我们选出的 $\overline{\mathcal{P}}$ 集合中，我们逼近可以选择的最优函数的过程造成了什么损失了。

3.0.2 步骤二：引理 C.1：逼近最优选择的过程中不会造成太大误差

这个引理直接研究 $\mathbb{E}[T \cdot \max_{p \in \overline{\mathcal{P}}} \mathrm{rev}(p) - \sum_{t=1}^{T} p_t\left(n_{i_t,p_t}\right) \leq \tilde{O}\left(m\sqrt{T}\right)$

4 买家类型被对手提前确定时的卖家定价算法

显然，作为买家方，我们可以针对卖家的定价方法针对性地做局，安排相应类型的买家使得卖家挣不到钱。例如可以先压价再买，或者先抬价再不买。这就需要卖家用一些方法来解决

在实际的设定中，买家的类型是在买卖之前由一个对手提前确定的。它可以针对卖家做局，但是不会根据卖家之后买卖时对定价函数的调整来动态在线做局。

作为卖家方，我们只有买家购买后，才能知道这个买家类型。不购买时，完全不知道对方类型，使得我们获取信息比较困难。而且买家类型可能被做局，故意误导卖家，需要算法具备抗干扰能力。

因此，我们的算法除了每一轮找出估计的收益最大的那个卖家定价函数，还需要利用已知信息在买家不购买数据导致我们无法获得信息的情况下编造一个信息，而且会在估计的收益中引入一个随机的扰动，即我们最终使用的估计值有一定随机性，以此来增加算法的灵活性。

4.1 算法 4：针对会提前做局但是不会动态调整的买家如何选出挣钱最多的定价

本算法的思路依然是计算出每个卖家的定价函数可能的收益，并选取最大收益的那个作为定价函数。

但是计算定价函数的收益不仅来源于历史数据，还来源于估计。

输入总买卖次数 T，我们选出的卖家定价函数 $\overline{\mathcal{P}}$，买家类型分布 P_i，控制随机扰动的强度的参数 θ。

对于 $\overline{\mathcal{P}}$ 中的每个函数 p，我们给它分配一个在未来不会变化的随机扰动 θ_p，这个 θ_p 被选择的概率为 $\theta e^{-\theta\theta_p}$，即按照概率密度函数 $\theta e^{-\theta x}$ 随机选择一个 θ_p。

我们需要一个指标来衡量每个定价函数的收益，因为收益越高代表至少我们觉得它越好，在算法中，我们使用符号 $r_{\tau(p)}$ 表示第 τ 轮我们通过实际买卖或者估计算出的卖家定价函数 p 的本轮收益。每一轮收益加起来就是这个定价函数的收益了。

接下来执行 T 轮以下步骤，不妨令本文为第 t 轮：

4.1.1 选择累计收益最高的卖家定价

我们将每一轮的收益 $r_{\tau(p)}$ 加起来，再加上之前指定的随机扰动 θ_p，得到估计的收益 $\sum_{\tau=1}^{T-1} r_{\tau(p)} + \theta_p$。选出结果最大的那个 p 然后用这个 p 这一轮买卖卖数据。

4.1.2 完成本轮的收益估计

下面我们将分知道本轮来了哪个买家或者不知道，来计算这个衡量每个定价函数的收益的指标 $r_{t(p)}$

如果这一轮买家买数据了，那么我们可以知道买家类型 i_t，买家内心估值 v_{i_t}，进而对我们选出的 $\overline{\mathcal{P}}$ 中的每一个函数 p，都可以算出这个买家会买的数据量 $n_{i_t,p}$，进而计算出如果本轮卖家选择这个定价函数，可以获得收益 $r_t(p) = p\big(n_{i_t,p}\big)$。

如果这轮买家没有买数据，那么我们自然不知道这个买家的类型，我们只能估计可能带来的收益。因为所有买家的估值曲线 v_i 都是已知信息，所以我们可以算出所有本轮因为定价 p_t 不会购买数据的买家的类型集合 S_t^c，然后对于其中的任何一个类型 i，算出其对每一个定价函数 p，会买 $n_{i,p}$ 个数据，产生 $p\big(n_{i,p}\big)$ 的收益，本轮函数 p 的估计收益 $r_t(p)$ 即为这些“可能存在但实际上没有”的收益之和。即 $r_t(p) = \sum_{i \in S_t^c} p\big(n_{i,p}\big)$

以上这两步需要对所有的卖家定价函数 $p \in \overline{\mathcal{P}}$ 进行。

4.2 定理 5.1：算法 4 给出了足够优秀的的卖家定价函数

Algorithm Specification

本文将算法拆分为两个部分，第一个部分是从无穷个卖家的定价函数中选取一些足够好的定价函数 $\overline{\mathcal{P}}$，第二个部分是根据不同的买卖规则，通过不断积累历史经验，在尽量挣钱的情况下找到 $\overline{\mathcal{P}}$ 中最好的那个 p。

1 从无穷个卖家的定价函数中选取一些足够好的定价函数

由于买家的定价函数通常很可能满足诸如平滑性、边际效用递减等限制，因此我们对于性质很好的买家函数，可以用不同的方法来选择更少但是损失数量级完全一样的函数们。

1.1 复习一下定义

1.1.1 m-step 函数

我们所有的三个算法都是要找出如 m-step 函数这个简单函数的集合

m-step ：一个函数 f 对于 $[N]$ 到 $[0,1]$ 的单调不减函数，满足至多有 m 个点 $n \in \{0,1,2,...,N\}, f(n+1)-f(n)>0$。

1.1.2 平滑性

对于买家的估值函数，相近的数据量效用不会差太多：

$\forall n, n' \in \{0,1,2,...,N\}, v_i(n+n')-v_i(n) \leq \frac{L}{N}n'$

1.1.3 边际效用递减

买家买的数据越多，数据越不值钱

$\exists J>0, \forall i \in \{0,1,2,...,m\}, \forall n \in \{0,1,2,...,N\}, v_i(n+1)-v_i(n) \leq \frac{J}{n}$

1.2 算法 1：估值函数仅符合单调递增时，如何选择卖家定价函数

输入误差范围 ε，买家种类 m

首先用等比数列 Z_i 对于卖家估值函数的值域作切分。

$Z_i = \left\{\varepsilon(1+\varepsilon)^i, \forall i = 0,1,2,...,\left\lceil \log_{1+\varepsilon}\frac{1}{\varepsilon}\right\rceil\right\}$

接着对每一个部分作插值，每个大块值域切分，进一步细化值域

$W_i = \left\{Z_{i-1}+Z_{i-1}\cdot\frac{\varepsilon k}{m} \mid k=1,2,3,...,\lceil(2+\varepsilon)m\rceil\right\}$

$W = \cup_{\{i=1\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$

然后输出 $\overline{\mathcal{P}}$ ，表示所有从 $\{0, 1, 2, ..., N\}$ 到 W 的 m-step 映射

1.3 算法 5 ：加上平滑性的限制后如何选择卖家定价函数

输入误差范围 ε，买家种类 m，Smoothness constant L（$\forall n, n' \in \{0, 1, 2, ..., N\}, v_i(n + n') - v_i(n) \leq \frac{L}{N} n'$ ）

首先用等比数列 Z_i 对于卖家估值函数的值域作切分。

$Z_i = \left\{ \varepsilon(1+\varepsilon)^i, \forall i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil \right\}$

接着对每一个部分作插值进一步细化值域

$W_i = \left\{ Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \mid k = 1, 2, 3, ..., \lceil (2+\varepsilon)m \rceil \right\}$

$W = \cup_{\{i=1\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$

以上步骤和算法 1 完全相同。

然后把 $[0, N]$ 划分为均匀网络，m-step 函数的跳跃点仅取这个划分中的点。

离散化间隔 $\delta = \left\lfloor \frac{\varepsilon N}{mL} \right\rfloor$表示划分的长度。

定义跳跃点可以取的点为 $N_s = \left\{ \delta k : k \in \left\{ 0, 1, 2, ..., \left\lceil \frac{N}{\delta} \right\rceil \right\} \right\}$。

然后输出 $\overline{\mathcal{P}}$ ，表示所有从 N_s 到 W 的 m-step 映射。

1.4 算法 2 ：加上边际效用递减的限制后如何选择卖家定价函数

需要输入边际收益递减常数 J，输入误差范围 ε，买家种类 m，

首先用等比数列 Z_i 对于卖家估值函数的值域作切分。

$Z_i = \left\{ \varepsilon(1+\varepsilon)^i, \forall i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil \right\}$

接着对每一个部分作插值进一步细化值域

$W_i = \left\{ Z_{i-1} + Z_{i-1} \cdot \frac{\varepsilon k}{m} \mid k = 1, 2, 3, ..., \lceil (2+\varepsilon)m \rceil \right\}$

以上步骤和算法 1 完全相同。

$W = \cup_{\{i=2\}}^{\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil} W_i$

对于定价函数的定义域 $[N]$ ，我们先用一个等比数列分割：

$Y_i = \left\lfloor \frac{2Jm}{\varepsilon^2} (1+\varepsilon^2)^i \right\rfloor \sim i = 0, 1, 2, ..., \left\lceil \log_{1+\varepsilon^2} \frac{N\varepsilon^2}{2Jm} \right\rceil$

对于每个小范围 $[Y_i, Y_{i+1})$，用步长 $Y_i \cdot \frac{\varepsilon^2}{2Jm}$ 进行插值

$Q_i = \left\{ Y_i + Y_i \cdot \frac{\varepsilon^2 k}{2Jm}, k = 0, 1, 2, 3, ..., \lfloor 2Jm \rfloor \right\}$

$Q = \cup_{\{i=1\}}^{\left\lceil \log_{1+\varepsilon^2} \frac{N\varepsilon^2}{2Jm} \right\rceil} Q_i$

定义可以取跳跃点的点为 $N_D = \left\{ 1, 2, ..., \left\lfloor \frac{2Jm}{\varepsilon^2} \right\rfloor \right\} \cup Q$

挑选出的卖家定价函数集合为 $\overline{\mathcal{P}}$ 表示所有只能在 N_D 跳变，定义域在 $[N]$ 且取值只能在 W 的单调不减 m-step 函数。

2 买家类型存在一个概率分布时的卖家定价算法

作为卖家方，我们回顾一下我们的挑战。首先我们知道买家有一个概率分布，但是我们不知道买家的概率分布是什么。知道了买家的概率分布我们就可以在之前选出的一系列定价函数 $\overline{\mathcal{P}}$ 中选出一个让我们挣钱最多的。

所以我们的算法任务是，根据之前来的买家的结果，估计出买家的分布，接着选出一个合适的定价函数来挣钱。

作为研究者，我们需要证明这个算法求出的函数是足够好的。

2.1 算法 3：针对有概率分布的买家如何选出挣钱最多的定价

买家类型的概率分布固定，但是卖家不知道这个具体的概率分布，只知道买家的种类数量和每种买家的估价函数，通过每轮选择计算出的“可能最大收益”（即收益的上置信界）最大的定价曲线。

这样既可以按照历史经验最大化本轮的收益，也可以逐渐根据历史经验算出哪个曲线是最好的，逐步逼近最优定价。

同时，算法中也近似的算出来买家种类的分布。

算法需要输入来购买的轮数 T，之前选出来的卖家比较好的定价函数集合 $\overline{\mathcal{P}}$。买方会选择一个购买商品的数量，使得此时自己获得最大的利润，即买方的估值函数减去卖方估值函数

第一轮先免费卖一次数据，显然这个类型为 i_1 的买家会买走所有 N 个数据点。

2 到 T 轮中的第 t 轮分别进行以下步骤：

2.1.1 用 UCB 法计算 $\overline{\mathcal{P}}$ 中每个卖家定价函数 p 的最大期望收益 $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$

计算步骤如下：

1. S_τ 表示“第 τ 轮有意向购买的买家编号”，是一个集合。含义是按照第 τ 轮卖家的历史定价函数 p_τ，会买数据的买家，他们的编号的集合。即如果卖家的函数上存在一点 n_i 使得 $v_i(n_i) \geq p_\tau(n_i)$，即如果本轮参与购买可以获益，那么这个买家的编号就会包含在 S_τ 中。
2. $T_{i,t}$ 表示“前 t 编号为 i 的买家本应该购买的次数”，即按照前 t 轮卖家的历史定价，如果类型为 i 的买家全去买了一遍会在几轮中购买。因为第一轮是白送的，所以买家至少可以在这一次购买中获益，所以如果每次购买都参加那么至少买这一次，即 $T_{i,t} \geq 1$。

3. $\overline{q}_{i,t}$ 表示"根据前 t 轮结果估计的类型 i 的买家的比例"。具体算法是实际上这 t 轮来购买数据的 i 类型买家数量，除以"前 t 轮编号为 i 的买家本应该购买的次数"。
4. $\hat{q}_{i,t}$ 表示"根据前 t 轮结果估计出的买家比例上界"，即 $\overline{q}_{i,t}$ 加上一个长得很像 UCB 经典公式的妙妙公式，该公式和总买卖轮次 T 和"本应该购买的次数"有关。公式形式为 $\hat{q}_{i,t} = \overline{q}_{i,t} + \sqrt{\frac{\log(T)}{T_{i,t}}}$
5. 计算"最大期望收益" $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$，即按照"根据前 t 轮结果估计出的买家比例上界"估计出收益的上界，方法是用$\hat{q}_{i,t}$ 这个"根据前 t 轮结果估计出的买家比例上界"计算收益的期望。 $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p) = \sum_{i=1}^{m}\left(\hat{q}_{i,t} \cdot p\left(n_{i,p}\right)\right)$

2.1.2 选出本轮采用的卖家定价函数 p

选出上一步算出的"最大期望收益" $\mathrm{UCB}_{\widehat{\mathrm{rev}}_{t-1}}(p)$ 最大的那个 p，这就是本轮卖家的定价 p_t

2.1.3 完成买卖

类型为 i_t 的买家会买走 n_{i_t,p_t} 个数据点，并付钱 $p_t\left(n_{i_t,p_t}\right)$。

其中 i_t 是从买家的分布中按照概率随机抽取的。

3 买家类型被对手提前确定时的卖家定价算法

显然，作为买家方，我们可以针对卖家的定价方法针对性地做局，安排相应类型的买家使得卖家挣不到钱。例如可以先压价再买，或者先抬价再不买。这就需要卖家用一些方法来解决

在实际的设定中，买家的类型是在买卖之前由一个对手提前确定的。它可以针对卖家做局，但是不会根据卖家之后买卖时对定价函数的调整来动态在线做局。

作为卖家方，我们只有买家购买后，才能知道这个买家类型。不购买时，完全不知道对方类型，使得我们获取信息比较困难。而且买家类型可能被做局，故意误导卖家，需要算法具备抗干扰能力。

因此，我们的算法除了每一轮找出估计的收益最大的那个卖家定价函数，还需要利用已知信息在买家不购买数据导致我们无法获得信息的情况下编造一个信息，而且会在估计的收益中引入一个随机的扰动，即我们最终使用的估计值有一定随机性，以此来增加算法的灵活性。

3.1 算法 4：针对会提前做局但是不会动态调整的买家如何选出挣钱最多的定价

本算法的思路依然是计算出每个卖家的定价函数可能的收益，并选取最大收益的那个作为定价函数。

但是计算定价函数的收益不仅来源于历史数据，还来源于估计。

输入总买卖次数 T，我们选出的卖家定价函数 $\overline{\mathcal{P}}$，买家类型分布 P_i，控制随机扰动的强度的参数 θ。

对于 $\overline{\mathcal{P}}$ 中的每个函数 p，我们给它分配一个在未来不会变化的随机扰动 θ_p，这个 θ_p 被选择的概率为 $\theta e^{-\theta\theta_p}$，即按照概率密度函数 $\theta e^{-\theta x}$ 随机选择一个 θ_p。

我们需要一个指标来衡量每个定价函数的收益，因为收益越高代表至少我们觉得它越好，在算法中，我们使用符号 $r_{\tau(p)}$ 表示第 τ 轮我们通过实际买卖或者估计算出的卖家定价函数 p 的本轮收益。每一轮收益加起来就是这个定价函数的收益了。

接下来执行 T 轮以下步骤，不妨令本文为第 t 轮:

3.1.1 选择累计收益最高的卖家定价

我们将每一轮的收益 $r_{\tau(p)}$ 加起来，再加上之前指定的随机扰动 θ_p，得到估计的收益 $\sum_{\tau=1}^{T-1} r_{\tau(p)} + \theta_p$。选出结果最大的那个 p 然后用这个 p 这一轮买卖卖数据。

3.1.2 完成本轮的收益估计

下面我们将分知道本轮来了哪个买家或者不知道，来计算这个衡量每个定价函数的收益的指标 $r_{t(p)}$

如果这一轮买家买数据了，那么我们可以知道买家类型 i_t，买家内心估值 v_{i_t}，进而对我们选出的 $\overline{\mathcal{P}}$ 中的每一个函数 p，都可以算出这个买家会买的数据量 $n_{i_t,p}$，进而计算出如果本轮卖家选择这个定价函数，可以获得收益 $r_t(p) = p\big(n_{i_t,p}\big)$。

如果这轮买家没有买数据，那么我们自然不知道这个买家的类型，我们只能估计可能带来的收益。因为所有买家的估值曲线 v_i 都是已知信息，所以我们可以算出所有本轮因为定价 p_t 不会购买数据的买家的类型集合 S_t^c，然后对于其中的任何一个类型 i，算出其对每一个定价函数 p，会买 $n_{i,p}$ 个数据，产生 $p\big(n_{i,p}\big)$ 的收益，本轮函数 p 的估计收益 $r_t(p)$ 即为这些“可能存在但实际上没有”的收益之和。即 $r_t(p) = \sum_{i\in S_t^c} p\big(n_{i,p}\big)$

以上这两步需要对所有的卖家定价函数 $p \in \overline{\mathcal{P}}$ 进行。

Experiments and Results

1 Algorithm Implementation

算法实现相对来说难度不大，只需根据伪代码进行实现即可.

定价空间的离散化（算法 1）:

```
def make_W(m, epsilon=0.1, begin=1):
    Z = [epsilon * (1 + epsilon) ** i for i in
range(math.ceil(math.log(1 / epsilon, 1 + epsilon)) + 1)]
    W = []
    for i in range(begin, len(Z)):
        for k in range(1, math.ceil((2+epsilon) * m) + 1):
            W.append(Z[i - 1] + Z[i - 1] * epsilon / m * k)
    return sorted(W)
```

数据空间离散化. 根据引理我们只需考虑所有的 “m-step” 价格曲线，因此我们可以用 m 个变化点的信息来表示价格曲线.下面依次为平滑条件下和递减回报条件下的离散化实现.

```
def discretize_smooth(N, m, epsilon, L):
    W = make_W(m, epsilon)
    delta = math.floor((epsilon * N) / (m * L))
    if delta == 0:
        delta = 1
    N_S = [delta * k for k in range(1, math.ceil(N / delta) + 1)]
    N_combs  = list(itertools.combinations(N_S, m))
    W_combs = list(itertools.combinations(W, m))
    P = []
    for n_comb, w_comb in itertools.product(N_combs, W_combs):
        P.append([np.array(list(n_comb)), np.array(list(w_comb))])
    return P
```

```
def discretize_diminishing(N, m, epsilon, J):
    W = make_W(m, epsilon, begin = 2)
    Y = [(2 * J * m) / (epsilon ** 2) * (1 + epsilon ** 2) ** i for i in
range(math.ceil(math.log(N * epsilon ** 2 / (2 * J * m), 1 + epsilon **
```

```
2)) + 1)]
    N_D = [k for k in range(1, math.ceil((2 * J * m) / (epsilon ** 2)) +
1)]
    for i in range(len(Y) - 1):
        for k in range(math.floor(2 * J * m)):
            q_i = math.floor(Y[i] + Y[i] * (epsilon ** 2) / (2 * J * m) *
k)
            if q_i not in N_D:
                N_D.append(q_i)
    N_combs = list(itertools.combinations(N_D, m))
    W_combs = list(itertools.combinations(W, m))
    P = []
    for n_comb, w_comb in itertools.product(N_combs, W_combs):
        P.append([np.array(list(n_comb)), np.array(list(w_comb))])
    return P
```

求解算法的实现. 随机（非对抗）情形:

```
def random_online_pricing(m, types, Time, Value, Price_curves):
    T_bound = np.ones(m)
    T_fact = np.zeros(m)
    T_fact[types[0]] = 1
    sum = 0
    records = [0]
    for time in range(1, Time):
        q = np.zeros(m)
        for idx in range(m):
            q[idx] = T_fact[idx] / T_bound[idx] +
math.sqrt(math.log(Time) / T_bound[idx])
        idx = types[time]
        max_rev = 0
        optimal_p = None
        for p in Price_curves:
            rev = 0
            for k in range(m):
                _, val = optimal_purchase(p, Value[k])
                stage = np.searchsorted(p[0], val, side="left")
                if stage >= len(p[0]):
                    stage = len(p[0]) - 1
                if val > 0:
                    rev += p[1][stage] * q[k]
            if rev > max_rev:
                max_rev = rev
                optimal_p = p
        a = optimal_purchase(optimal_p, Value[0])
        a = optimal_purchase(optimal_p, Value[1])
```

```
        sum += max_rev
        records.append(optimal_p)
        for k in range(m):
            _, val = optimal_purchase(optimal_p, Value[k])
            if val > 0:
                T_bound[k] += 1
                if k == idx:
                    T_fact[k] += 1

    return sum, records
```

对抗情形：

```
def adversarial_online_pricing(m, types, Time, Value, Price_curves,
theta=10):
    theta_p = [np.random.exponential(scale=1/theta) for _ in Price_curves]
    r_sum = np.zeros(len(Price_curves))
    sum = 0
    records = []
    for time in range(Time):
        optimal_p = Price_curves[np.argmax(r_sum + theta_p)]
        idx = types[time]
        _, val = optimal_purchase(optimal_p, Value[idx])

        for k, p in enumerate(Price_curves):
            stage = 0
            if val > 0:
                # print(p, Value[idx])
                stage = np.searchsorted(p[0], optimal_purchase(p,
Value[idx])[1], side="left")
                if stage >= len(p[0]):
                    stage = len(p[0]) - 1
                r_sum[k] += p[1][stage]
            else:
                for j in range(m):
                    if optimal_purchase(p, Value[j])[1] == 0:
                        stage = np.searchsorted(p[0], optimal_purchase(p,
Value[j])[1], side="left")
                        if stage >= len(p[0]):
                            stage = len(p[0]) - 1
                        r_sum[k] += p[1][stage]
        stage = np.searchsorted(optimal_p[0], val, side="left")
        if stage >= len(optimal_p[0]):
            stage = len(optimal_p[0]) - 1
        sum += optimal_p[1][stage]
```

```
        records.append(optimal_p)
    return sum, records
```

其中 optimal_purchase 函数用于计算最优购买.

```
def optimal_purchase(P, V):
    max_rev, val= 0, 0
    for i in range(1, len(V)):
        stage = np.searchsorted(P[0], i, side="left")
        if stage >= len(P[0]):
            stage = len(P[0]) - 1
        rev = V[i] - P[1][stage]
        if rev > max_rev:
            max_rev = rev
            val = i
    return max_rev, val
```

2 Test Case Generation

3 Results and Analysis

Conclusion