上海交通大学安泰经济与管理学院

BUSS3620 人工智能导论

第一学期, 2023-24

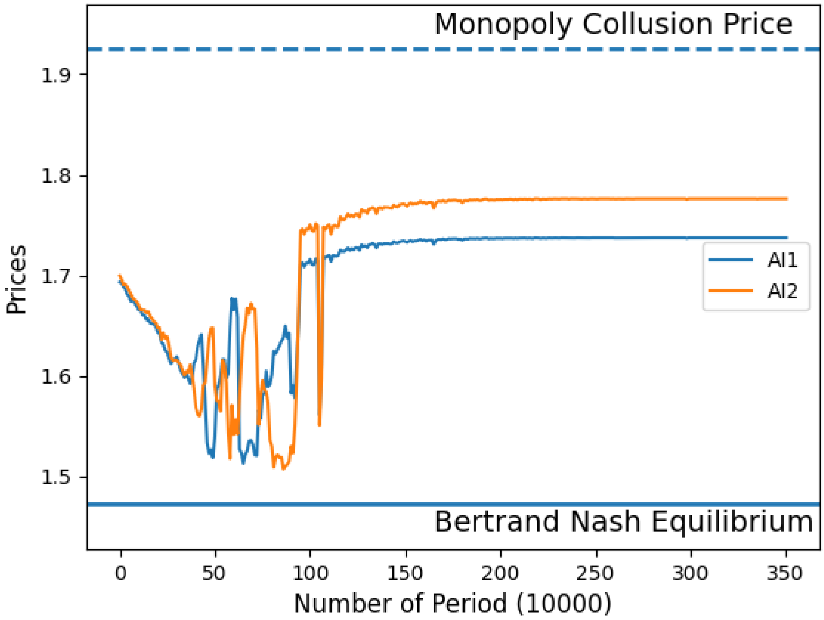
Project #6. 算法共谋

刘佳璐 助理教授

算法共谋

本次项目旨在复现发表在《American Economic Review》中的论文《Artificial Intelligence, Algorithmic Pricing, and Collusion》里的基础结果（Baseline）。这篇论文探讨了一个令人担忧的问题：在企业越来越依赖 AI 算法进行商品定价的情况下，这些算法会自主学习并形成“共谋”，集体抬高商品价格，损害消费者利益。论文作者之后又在《Science》发表了介绍算法共谋的总结性工作。

```
$ python pricing.py
t=3500000Converged!
Done training
AI1 chooses price 1.7377012974373758
AI2 chooses price 1.776448805350559
```



背景介绍

《Artificial Intelligence, Algorithmic Pricing, and Collusion》这篇论文发现，当 AI 算法自主学习定价时，竟然能够在没有任何人类干预或明确沟通的情况下学会“勾结”——像偷偷结盟的商人一样，通过抬高价格来赚更多钱。为了研究这个现象，作者使用 Q 学习（Q-learning）模拟多轮定价游戏，观察算法在不同市场条件下的定价行为。

这篇文章的定价游戏是经济学中经典的伯特兰定价模型（Bertrand Game）。n个企业（为了简化，假设$n = 2$）各生产一种商品$i = 1,2$，市场上还有一个外部商品（outside goods），2 家企业同时决定商品的价格p_i，商品的销量q_i完全取决于商品的价格p_i，如下：

$$q_i = \frac{e^{\frac{a_i - p_i}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_j}{\mu}} + e^{\frac{a_0}{\mu}}}.$$

其中a_i衡量商品的质量差异，μ为顾客的品味差异。企业的利润函数为$\pi_i = (p_i - c_i)q_i$，c_i是企业生产商品的边际成本。这个定价模型理论上的纳什均衡(Bertrand Nash Equilibrium)为$p_{nash} = c_i + \frac{\mu}{1-q_i}$（证明见本文档证明部分）。假设两个企业合谋，共同制定垄断价格，那么最极端的情况下，每个企业都会将两个企业的总利润最大化，这样又可以得到一个的垄断价格(Monopoly Collusion Price)为p_{max}（证明见本文档证明部分）。

举例而言，假设两个公司是非常类似的公司，生产非常类似的产品，我们假定一些模型参数为$c_i = 1, a_i - c_i = 1, a_0 = 0, \mu = 0.25$（论文中 Baseline 参数），那么①如果两个公司只关注各自的利润最大化，他们各自对产品的定价应为 1.47；②如果两个公司形成合谋垄断情况，关注共同利润最大化，他们各自对产品的定价应为 1.92。这里可以看到，如果 AI（Q-learning）代替公司进行定价，且最终 AI 的定价偏离了 1.47，在 1.47-1.92 之间，那么就反映出 AI 一定程度上学会“勾结”，通过集体抬高商品价格（往垄断的方向发展），损害消费者利益。

在课上 Q 学习算法中，我们尝试学习每一个`(state, action)`的 Q 值，帮助智能体进行动作选择。那我们要如何表示定价游戏中的状态和动作？论文作者假设`action`是 AI 每一轮的出价，而`state`是两个 AI 前一轮的出价。一般强化学习算法要求智能体可选择的动作是离散的，但在这个定价游戏中，AI 可选择的定价是连续的。因此，作者将连续的定价空间离散化，在定价游戏中，假设 AI 只能选择$m = 15$个价格，分别为区间$[p_{nash} - \xi \times (p_{max} - p_{nash}), p_{nash} + \xi \times (p_{max} - p_{nash})]$内的 15 个等间距点。以论文中 Baseline 参数($\xi = 0.1$)为例，AI 可选择的价格为`[1.42772123,1.46646874,1.50521625,1.54396376,1.58271127,1.62145877,1.66020628,1.69895379,1.7377013,1.77644881,1.81519631,1.85394382,1.89269133,1.93143884,1.97018634]`。为了提高算法效率，我们可以假设 AI 可选择的动作为`0-14`之间的数字，即`[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]`。然后将数字与价格对应起来，即`action=0`时意味着 AI 选择的价格是价格区间的第一个位置的价格`1.42772123`。这样我们也可以用正整数来表示`state`，比如`state=(1,2)`意味着上一轮两个 AI 的出价为`(1.46646874,1.50521625)`。

回想一下，Q 学习的关键公式如下。每当我们处于某个状态`s`并采取某个行动`a`时，我们都可以根据以下公式更新 Q 值`Q(s, a)`: `Q(s, a) <- (1 - alpha) * Q(s, a) + alpha * (reward + gamma * best future value)`。这篇论文的 Baseline 假设`alpha=0.15`, `gamma=0.95`。

作者还采用了ϵ贪婪（ϵ-greedy）算法。即智能体有ϵ的概率随机选择一个动作，有$1-\epsilon$的概率选择最佳动作。在这篇论文里，作者假设每一轮次的$\epsilon_t = \mathrm{e}^{-\beta t}$，其中$\beta = 4 \times 10^{-6}$，即$\epsilon$并非一个固定不变的值。

那么什么时候我们可以说 AI 学好了呢？作者认为 AI 如果在最新的`ct=100000`个轮次中都选择同样的动作时，则认为 AI 学好了，收敛了（converge），程序可以停止了。如果一直没有收敛，那么运行`tmax=10000000`轮次后，即可停止。

开始

- 从课程中心平台 Canvas 上下载`week10 强化学习`单元中的`week10_project.zip`并且解压缩
- 当处于本项目文件所在的工作目录中时，在终端上运行`pip3 install -r requirements.txt`用来安装这次项目需要的 Python 包。

理解项目的相关文件

这个项目主要包含一个文件：`pricing.py`。

打开`pricing.py`。此文件中定义了两个类（`PricingGame`和`AI`）以及三个函数（`train`、`draw`、`main`）。`PricingGame`、`draw`、`main`已经写好，而类`AI`及`train`函数需要你来完成。

首先看一下类 `PricingGame`，这个类包括定价游戏的参数 `a` （商品质量差异的列表）,`c`（商品边际成本列表）,`mu`（顾客品味偏好)。这个类还包含根据价格计算销量的函数 `demand`，计算纳什均衡p_{nash}的函数 `bertrand_nash_equilibrium`，计算垄断价格p_{max}的函数 `monopoly_price`，以及 `compute_profits` 函数，能够给定两个 AI 出价 p，计算他们各自的利润。因为游戏涉及到 2 个玩家，所以以上函数返回的都是两个玩家分别对应的数值。注意，函数 `demand`、`bertrand_nash_equilibrium`、以及 `monopoly_price` 均为 `classmethod`，这意味着你可以直接调用函数，无需创建一个实体去调用函数。例如，你可以通过 `PricingGame.bertrand_nash_equilibrium(a, c, mu)`直接计算出给定 `a,c,mu` 的情况下的纳什均衡p_{nash}。

接下来看一下类 `AI`，它定义了我们参与定价游戏的 `AI`。在`__init__`函数中，我们定义了 AI 的相关参数并提供了默认值（论文 Basline 的参数值）：首先它包含定价游戏的参数 `a,c,mu`。其次还包含 Q 函数中的学习速率 `alpha`，折现系数 `gamma`。与 AI 动作空间相关的参数 `ksi`，`m`，与ϵ贪婪算法相关的 `beta`，与收敛相关的参数（`ct`、`tmax`、`count`），`ct` 与 `tmax` 在背景介绍部分已经讲解，`count` 记录 AI 已经在多少个轮次中选择同样的动作。除了这些参数以外，我们还将所有 Q 值初始化为 0，为了加快运行速度，我们将之储存成列表，大小为 `15 x 15 x 15`。即对每一个`(state, action)`你可以通过 `self.Q[state[0]][state[1]][action]`获得对应的 Q 值，注意这里无论是 `state` 中的元素还是 `action`，都是 `0-14` 之间的正整数，而非包含小数的价格。同时类 AI 将可选择的价格记录在 `self.prices`，这个列表里面每个元素为可选择的价格（小数）。`self.action` 记录了 AI 训练过程中所选的每一个 `action`。类 `AI` 中的其他函数需要你来编写。

`main` 函数已经为你写好。首先程序会初始化一个定价游戏 `game`，初始化两个 `AI`，分别为 `ai1`，`ai2`。然后两个 AI 分别随机选择 1 个动作 `a1`,`a2`，此时为 `t=0` 时刻。接下来两个 AI 会不停重复玩定价游戏，直到两个 AI 的动作收敛为止，此时程序会画出 AI 的价格图并输出最终选择的价格。注意本项目中 AI 最终输出的价格并非固定的，每次重新训练后的最终价格可能都不太一样，在论文中，作者提到最终的价格的均值在 1.8 左右，同时算法也不保证收敛，会存在无法收敛的情况。

`draw` 函数已经写好根据两个 AI 记录的动作画出 AI 的价格选择图。

类 `AI` 中剩余的函数，以及 `train` 函数，将留给同学们你来完成。`Week10_project.zip` 中还包含两篇论文的 pdf 文件，以及 `autograde` 文件夹，里面包含测试代码的相关文件。

要求

类 `AI` 中的 `available_prices` 函数

- 输入：无
- 功能：返回一个 AI 的可选的价格列表
    - AI 只能选择 15 个价格，分别为区间$[p_{nash} - \xi \times (p_{max} - p_{nash}), p_{nash} + \xi \times (p_{max} - p_{nash})]$内的 15 个等间距点
    - 你可以根据给定的 `a,c,mu` 计算对应的p_{nash}, p_{max}，再找到区间内的 15 个等间距价格
- 输出：包含 15 个元素的列表

`action_to_price` 函数

- 输入：一个 `action`（`0-14` 之间的正整数）
- 功能：给定 `action`，返回该 `action` 对应的价格
    - `action` 代表的是 AI 可选的价格列表中价格的位置
- 输出：一个价格

`update` 函数

- 输入：`old_state`（上一轮 2 个 AI 的 `action`），`action`（本轮本 AI 的 `action`），`new_state`（本轮 2 个 AI 的 `action`），`reward`（本轮本 AI 的利润）
- 功能：根据 Q 学习公式更新 Q 值
  - Q 学习公式为 `Q(s, a) <- (1 - alpha) * Q(s, a) + alpha * (reward + gamma * best future value)`
- 输出：无

`choose_action` 函数

- 输入：`state`（上一轮 2 个 AI 的 `action`），`t`（训练的轮次）
- 功能：采用 ϵ 贪婪算法选择动作
  - AI 有ϵ的概率随机选择一个动作，有$1-\epsilon$的概率选择最佳动作
  - t 时刻的$\epsilon_t = e^{-\beta t}$
  - 如果多个动作具有相同的 Q 值，则其中任何一个选项都是可接受的返回值
  - 你需要将 AI 选择的动作添加到 `self.action` 列表中
- 输出：一个动作 `action`（`0-14` 之间的正整数）

`check_convergence` 函数

- 输入：无
- 功能：判断 AI 是否收敛
  - AI 如果在最新的 `ct=100000` 个轮次中都选择同样的动作时，则认为 AI 学好了，收敛了
  - 你可以查看本轮 AI 选择的动作和上一轮是否一样，如果一样的话，可以在 `self.count` 中加 1，如果不一样，将 `self.count` 重置为 `0`
- 输出：`True` 或 `False`
  - 如果 AI 收敛返回 `True`，不收敛但是超过最大轮次 `tmax`，也返回 `True`，否则返回 `False`

`train` 函数

- 输入：`state`（上一轮 2 个 AI 的 `action`），`t`（训练的轮次）
- 功能：两个 AI 分别选择行动，获得相应的利润，更新自己的 Q 值
- 输出：两个 AI 选择的动作 `a1,a2`

你不应该修改 `pricing.py` 中已经写好的其他部分，但是你可以添加新的函数。如果你熟悉 `numpy`，你也可以使用它，但是你不可以使用其他的第三方包。

测试代码

- 你可以使用代码 `pytest autograde/autograde.py --tb=no` 自行测试自己的代码是否满足要求。您需要安装 `requirements.txt` 中的 `pytest` 包。
- 请先确保你的程序能够成功运行并输出结果。请确保你的工作目录中包含 `pricing.py`。

文献

Calvano, Emilio, Giacomo Calzolari, Vincenzo Denicolò, and Sergio Pastorello. 2020. "Artificial Intelligence, Algorithmic Pricing, and Collusion." American Economic Review, 110 (10): 3267–97.

Calvano, Emilio, Giacomo Calzolari, Vincenzo Denicolò, Joseph E. Harrington Jr., and Sergio Pastorello. 2020. "Protecting consumers from collusive prices due to AI." Science 370,1040-1042.

证明

- 伯特兰-纳什均衡

  步骤 1：定义需求函数

  对于每个产品 $i = 1,2$，需求为

$$q_i = \frac{e^{\frac{a_i - p_i}{\mu}}}{\sum_{j=1}^{n} e^{\frac{a_j - p_j}{\mu}} + e^{\frac{a_0}{\mu}}}$$

  步骤 2：推导利润函数

  企业 i 的利润为：

$$\pi_i = (p_i - c_i)q_i$$

  步骤 3：计算一阶倒数（FOC）

  为了找到伯特兰-纳什均衡，每个企业在给定竞争对手价格的情况下，最大化其自身利润，因此需要对π_i关于p_i求导:

$$\frac{\partial \pi_i}{\partial p_i} = (p_i - c_i)\frac{\partial q_i}{\partial p_i} + q_i = (p_i - c_i)(-\frac{1}{\mu}q_i(1 - q_i)) + q_i$$

  步骤 4：将一阶倒数设为 0，解出p_i，得到

$$p_i = c_i + \frac{\mu}{1 - q_i}$$

- 最大化总利润的垄断价格

  步骤 1：定义总利润

  垄断者最大化总利润

$$\pi = \pi_1 + \pi_2 = (p_1 - c_1)q_1 + (p_2 - c_2)q_2$$

  步骤 2：计算一阶倒数（FOC）

  对π_i关于p_i求导：

$$\frac{\partial \pi}{\partial p_1} = (p_1 - c_1)\frac{\partial q_1}{\partial p_1} + q_1 + (p_2 - c_2)\frac{\partial q_2}{\partial p_1}$$

$$\frac{\partial \pi}{\partial p_2} = (p_2 - c_2)\frac{\partial q_2}{\partial p_2} + q_2 + (p_1 - c_1)\frac{\partial q_1}{\partial p_2}$$

  步骤 3：将一阶倒数设为 0，得到

$$-(p_1 - c_1)(1 - q_1) + (p_2 - c_2)q_2 + \mu = 0$$
$$-(p_2 - c_2)(1 - q_2) + (p_1 - c_1)q_1 + \mu = 0$$

  所以商品 1 和商品 2 的垄断价格可通过解以上方程组求得。