推荐系统应用多臂老虎机算法

龙旷飞/李泽浩

计算机科学与技术专业

华东师范大学

指导老师：张伟

目录

1. 推荐系统简介与新内容

随着今天越来越多的直接面向消费者（DTC）平台的选择，大多数消费者无法订阅所有平台。订阅/购买决定是由内容和用户体验共同驱动的。今天的消费者在考虑、购买和接触内容时，期望获得实时、精心策划的体验。无论是提高点击率、增加观看次数、观看时间、订阅或购买优质内容，媒体公司都在努力寻找方法，以提供更好的客户体验并扩大盈利能力。

推荐系统是实现这些目标的一个重要工具。DTC 平台提供的推荐可以最大限度地发挥深度内容目录的价值，在消费者观看了最初将他们带到平台的内容之后，还可以保持他们的参与。例如，视频点播（VOD）平台的良好推荐可以通过在基于消费者行为的推荐中浮现长尾内容而增加收入。我们首先回顾目前使用的常见的推荐系统的种类。然后，我们深入研究了这个领域中的一些最新发展。

1.1. 常见推荐系统

常见系统可以被归类为基于内容的过滤或协作式过滤。基于内容的过滤是最简单的系统之一，但有时仍然是有用的。它是基于明确或隐含地提供的已知的用户偏好，以及关于项目特征的数据（比如项目所属的类别）。虽然这些系统很容易实现，但它们的建议往往让人感觉是静态的，而且很难处理那些偏好未知的新用户。

协同过滤是基于（用户、项目、评级）给定特征的。因此，与基于内容的过滤不同，它利用了其他用户的经验。亚马逊公司是这种方法的先驱，并发表了一篇早期的论文，后来获得了 IEEE 颁发的最经得起 时间考验的论文。协同过滤的主要概念是，具有相似品味的用户（基于观察到的用户与物品的互动）更有可能与他们以前没有见过的物品产生相似的互动。

与基于内容的过滤相比，协同过滤为多样性（推荐项目的不同程度）、偶然性（衡量成功或相关推荐的随机程度）和新颖性（推荐项目对用户的未知程度）提供了更好的结果。然而，协同过滤的计算成本较高，实施和管理起来也更加复杂和昂贵。尽管一些用于协同过滤的算法比其他算法更轻便。协同过滤也有一个冷启动问题，因为如果没有大量的交互数据来训练模型，它很难推荐新的项目。

除了这两类经典的推荐系统外，各种神经网络架构在推荐系统中也很常见。 有些实现了一种协作过滤的形式。其他的推荐系统则扩展到处理时间性数据，以便根据反映用户兴趣演变的用户行为序列来进行推荐。这些系统最初是基于各种递归神经网络（RNN）的。现在，它们利用基于 Transformer 的模型，通过自我关注来学习用户行为序列中的项目之间的依赖关系。

1.2. 推荐系统中的新发现

在过去的几年里，研究人员已经尝试了许多新的推荐系统方法。事实上，有这么多的方法，不能在此一一介绍。其中，混合系统越来越受欢迎。其中一些较新的方法并不相互排斥，可以与彼此或早期的技术相结合。一个例子是亚马逊 Personalize，一个完全管理的个性化推荐服务。在 Amazon Personalize 中，用户个性化的首选算法结合了较新的基于 Bandit 的方法和基于 AWS 最近论文的 Hierarchical RNN。Bandit 也是我们本次期末项目的研究方向。

一个活跃的研究领域是纳入基于 Bandit 算法的推荐系统。 Bandit 算法是强化学习（RL）的一种形式，试图在探索新的可能性和利用已经发现的有利可图的可能性之间取得平衡。 它们经常被用作静态 A/B test 的替代品：一个关键的优势是它们能够实时适应。 这可以帮助克服冷启动的问题。

在推荐系统的背景下，Bandit 算法现在有很多应用，并已被整合到产业系统中，如 Amazon Personalize，它有效地将 RNN 与 Bandit 相结合，提供更准确的用户建模和有效的探索。 事实上，Bandit 算法可以用来根据用户对每个系统提供的不同建议的反应，在几个推荐系统之间进行实时选择。Bandit 算法的一个越来越重要的应用是在考虑到与用户满意度相关的多个目标和指标的系统中的情况。例如，在一个音乐内容推荐系统中，一个额外的目标可能是为默默无名

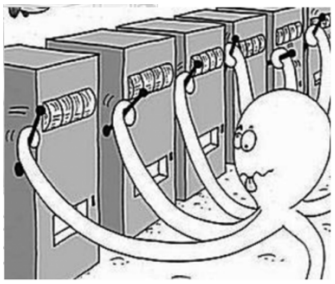
的艺术家和内容提供 "公平"，确保他们至少得到一些推荐。这种方法已经被 Spotify 等内容提供商研究过了，Spotify 的一位研究人员在一个公开的演讲中讨论过这个问题。

2. Bandit 算法

2.1. Bandit 起源

Bandit 算法来源于历史悠久的赌博学，它要解决的问题是这样的：一个以利润最大化为目标的风投公司，该公司面临着一个两难的选择。何时投资于已经成功的公司，何时投资于尚未成功但有巨大潜力的公司。 投资学告诉我们：回报总是伴随着风险。一个成功的风险投资家必须处理好这种探索和开发的权衡：过多的探索意味着无法获得更高的回报，而过多的开发则意味着错过获得更高回报的机会。更通俗的讲就是，一个赌徒，要去摇老虎机，走进赌场一看，一排老虎机，外表一模一样，但是每个老虎机吐钱的概率可不一样，他不知道每个老虎机吐钱的概率分布是什么，那么每次该选择哪个老虎机可以做到最大化收益。

在现实生活和商业中，我们都面临着这种两难境地，没有正确的答案来教你如何做，可能是因为我们对世界的理解还不够清晰。然而，在数学领域，这个问题已经被研究过，被称为多臂老虎机问题 (Multi-armed bandit problem, K-armed bandit problem, MAB)，简称 MAB 问题，也被称为顺序资源分配问题。它被广泛用于广告推荐系统、源路由和棋盘游戏。



2.2. Bandit 算法与推荐系统

在推荐系统领域里，有两个比较经典的问题常被人提起，一个是 EE 问题，另一个是用户冷启动问题。

EE 问题，又叫 exploit – explore 问题。exploit 就是：对用户比较确定的兴趣，当然要利用开采迎合，好比说已经挣到的钱，当然要花；explore 就是：光对着用户已知的兴趣使用，用户很快会腻，所以要不断探索用户新的兴趣才行，这就好比虽然有一点钱可以花了，但是还得继续搬砖挣钱，不然花完了就得喝西北风。 用户冷启动问题，也就是面对新用户时，如何能够通过若干次实验，猜出用户的大致兴趣。EE 问题涉及到平衡准确和多样，而冷启动问题涉及到产品算法运营等一系列。Bandit 算法是一种简单的在线学习算法，常常用于尝试解决这两个问题。

这两个问题本质上都是如何选择用户感兴趣的主题进行推荐，比较符合 Bandit 算法背后的 MAB 问题。

比如，用 Bandit 算法解决冷启动的大致思路如下：用分类或者 Topic 来表示每个用户兴趣，也就是 MAB 问题中的臂（Arm），我们可以通过几次试验，来刻画出新用户心目中对每个 Topic 的感兴趣概率。这里，如果用户对某个 Topic 感兴趣（提供了显式反馈或隐式反馈），就表示我们得到了收益，如果推给了它不感兴趣的 Topic，推荐系统就表示很遗憾（regret）了。如此经历选择-观察-更新-选择的循环，理论上是越来越逼近用户真正感兴趣的 Topic 的。

2.3. Bandit 算法与遗憾

现在来介绍一下 Bandit 算法怎么解决这类问题的。Bandit 算法需要量化一个核心问题：错误的选择到底有多大的遗憾？首先介绍一个概念，叫做累积遗憾（regret）：

$$R_T = \sum_{i=1}^{T}(w_{opt} - w_{B(i)}) = Tw^* - \sum_{i=1}^{T} w_{B(i)}$$

这个公式就是计算 Bandit 算法的累积遗憾，解释一下：

首先，这里我们讨论的每个臂的收益非 0 即 1，也就是伯努利收益。然后，每次选择后，计算和最佳的选择差了多少，然后把差距累加起来就是总的遗憾。$w_{B(i)}$是第 i 次试验时被选中臂的期望收益， w^*是所有臂中的最佳那个，如果上帝提前告诉你，我们当然每次试验都选它，问题是上帝不告诉你，所以就有了 Bandit 算法。

这个公式可以用来对比不同 Bandit 算法的效果：对同样的多臂问题，用不同的 Bandit 算法试验相同次数，看看谁的 regret 增长得慢。接下来介绍不同的 Bandit 算法。

2.4. 常用的 Bandit 算法

2.4.1. Thompson Sampling 算法

Thompson sampling 算法简单实用。简单介绍一下它的原理，要点如下：

· 假设每个臂是否产生收益，其背后有一个概率分布，产生收益的概率为 p。

· 我们不断实验，去估计一个置信度较高的概率 p 的概率分布就能近似解决这个问题了。

· 估计概率 p 的概率分布的方法是假设概率 p 的概率分布符合 beta(wins, lose)分布，它有两个参数: wins, lose。

· 每个臂都维护一个 beta 分布的参数。每次试验后，选中一个臂，摇一下，有收益则该臂的 wins 增加 1，否则该臂的 lose 增加 1。

· 每次选择臂的方式是用每个臂现有的 beta 分布产生一个随机数 b，选择所有臂产生的随机数中最大的那个臂去摇。

2.4.2. UCB 算法

UCB 算法全称是 Upper Confidence Bound（置信区间上界），它的算法步骤如下：

· 初始化：先对每一个臂都试一遍。

· 按照如下公式计算每个臂的分数，然后选择分数最大的臂作为选择：

$$\bar{x}_j(t) + \sqrt{\frac{2\ln t}{T_{j,t}}}$$

· 观察选择结果，更新 t 和 $T_{j,t}$，其中加号前面是这个臂到目前的收益均值，后面的叫做 bonus，本质上是均值的标准差，t 是目前的试验次数，$T_{j,t}$ 是这个臂被试次数。

这个公式反映了一个特点：均值越大，标准差越小，被选中的概率会越来越大，同时哪些被选次数较少的臂也会得到实验的机会。

2.4.3. ε – Greedy 算法

· 选一个（0，1）之间较小的数作为 ε。

· 每次以概率 ε 做一件事：所有臂中随机选择一个。

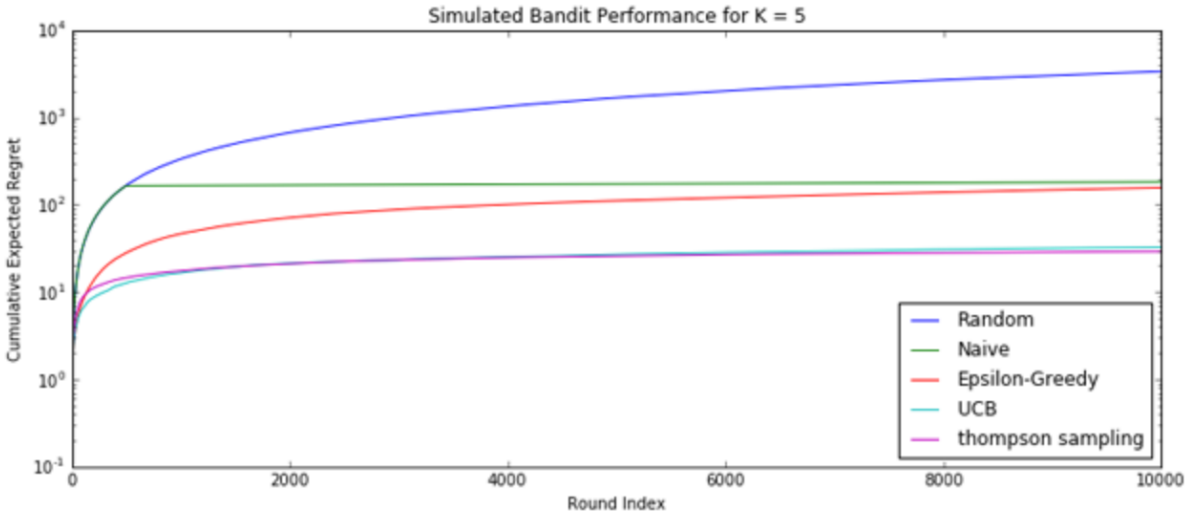
· 每次以概率 1 – ε 选择平均收益最大的那个臂。

ε的值可以控制对 Exploit 和 Explore 的偏好程度。越接近 0，越保守。

2.4.4. 朴素 Bandit 算法

最朴素的 Bandit 算法就是：先随机试若干次，计算每个臂的平均收益，一直选均值最大那个臂。这个算法是人类在实际中最常采用的，不可否认，它还是比随机乱猜要好。

这几个著名的算法，经过我们查阅资料及实验，得到性能对比结果图如下：



3. LinUCB 算法与改进算法

3.1. 回顾 UCB 算法

UCB 算法在做 EE（Exploit-Explore）的时候表现不错，但它是上下文无关（context free）的 Bandit 算法，它只管埋头干活，根本不观察一下面对的都是些什么特点的 arm，下次遇到相似特点但不一样的 arm 也帮不上什么忙。

UCB 解决 Multi-armed bandit 问题的思路是：用置信区间。置信区间可以简单地理解为不确定性的程度，区间越宽，越不确定。每个 item 的回报均值都有个置信区间，随着试验次数增加，置信区间会变窄（逐渐确定了到底回报丰厚还是可怜）。每次选择前，都根据已经试验的结果重新估计每个 Item 的均值及置信区间。选择置信区间上限最大的那个 Item。

· 如果 Item 置信区间很宽（被选次数很少并不确定），那么它会倾向于被多次选择，这个是算法冒风险的部分。

· 如果 Item 置信区间很窄（被选次数很多，比较确定其权衡好坏了），那么均值达到倾向于被多次选择，这个是算法保守稳妥的部分。

· UCB 算法中选择置信区间的上界排序时，是一种乐观的算法。UCB 算法中选择置信区间的下界排序时，是一种悲观保守的算法。

3.2. UCB 算法加入特征信息

单纯的老虎机回报情况就是老虎机自己内部决定的，而在广告推荐领域，一个选择的回报，是由 User 和 Item 一起决定的，如果我们能用 Feature 来刻画 User 和 Item，在每次选择 Item 之前，通过 Feature 预估每一个 arm（item）的期望回报及置信区间，选择的收益就可以通过 Feature 泛化到不同的 Item 上。

为 UCB 算法插上了特征的翅膀，这就是 LinUCB 的内涵。

LinUCB 算法做了一个假设：一个 Item 被选择后推送给一个 User，其回报和相关 Feature 成线性关系，这里的相关 Feature 就是 context，也是实际项目中发挥空间最大的部分。

于是试验过程就变成：用 User 和 Item 的特征预估回报及其置信区间，选择置信区间上界最大的 Item 推荐，观察回报后更新线性关系的参数，以此达到实验学习的目的。

$$E[r_{t,a}|x_{t,a}] = x_{t,a}^T \theta_a^*$$

其中部分参数：

· a = arm = 被推荐的实例。

· t = trail = 尝试的标记。

· $r_{t,a}$ = 实例 a 在第 t 次的 reward 回报。

· $x_{t,a}$ = 在选择实例 a 在第 t 次的时候描述用户和实例的共有特征（feature）。

· θ_a^* = 对特定 arm a 的未知系数向量。

LinUCB 基本算法描述，伪代码如下：

Inputs: $\alpha \in \mathbb{R}_+$
for $t = 1, 2, 3, \ldots, T$ **do**
  Observe features of all arms $a \in \mathcal{A}_t$: $\mathbf{x}_{t,a} \in \mathbb{R}^d$
  for all $a \in \mathcal{A}_t$ **do**
    if a is new **then**
      $\mathbf{A}_a \leftarrow \mathbf{I}_d$ (d-dimensional identity matrix)
      $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ (d-dimensional zero vector)
    end if
    $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\mathbf{b}_a$
    $p_{t,a} \leftarrow \hat{\boldsymbol{\theta}}_a^\top \mathbf{x}_{t,a} + \alpha\sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}_a^{-1}\mathbf{x}_{t,a}}$
  end for
  Choose arm $a_t = \arg\max_{a\in\mathcal{A}_t} p_{t,a}$ with ties broken arbitrarily, and observe a real-valued payoff r_t
  $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
  $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
end for

3.3. LinUCB 算法详情

```
class LinUCB:
    def __init__(self):
        self.alpha = 0.25
        self.r1 = 1
        self.r0 = 0
        # 用户特征的维度 d
        self.d = 6
        # Aa : 各个 a 的计算矩阵，维度为 d*d
        self.Aa = {}
        # AaI : Aa 矩阵的逆
        self.AaI = {}
        # ba : 计算向量，维度为 d*1
        self.ba = {}

        self.a_max = 0
        self.theta = {}

        self.x = None
        self.xT = None
        # linUCB
```

```
21.
22.     def set_articles(self, art):
23.         # 初始化矩阵/向量 Aa, Ba, ba
24.         for key in art:
25.             self.Aa[key] = np.identity(self.d)
26.             self.ba[key] = np.zeros((self.d, 1))
27.             self.AaI[key] = np.identity(self.d)
28.             self.theta[key] = np.zeros((self.d, 1))
29.     # 更新参数时不传入更新哪个 arm，在上一次 recommend 的时候缓存了被选的那个 arm，此处不用传入
30.     # 另外，update 操作不用阻塞 recommend，可以异步执行
31.     def update(self, reward):
32.         if reward == -1:
33.             pass
34.         elif reward == 1 or reward == 0:
35.             if reward == 1:
36.                 r = self.r1
37.             else:
38.                 r = self.r0
39.             self.Aa[self.a_max] += np.dot(self.x, self.xT)
40.             self.ba[self.a_max] += r * self.x
41.             self.AaI[self.a_max] = linalg.solve(self.Aa[self.a_max], np.identity(self.d))
42.             self.theta[self.a_max] = np.dot(self.AaI[self.a_max], self.ba[self.a_max])
43.         else:
44.         # error
45.             pass
46.         # 预估每个 arm 的回报期望及置信区间
47.     def recommend(self, timestamp, user_features, articles):
48.         xaT = np.array([user_features])
49.         xa = np.transpose(xaT)
50.         art_max = -1
51.         old_pa = 0
52.
53.         # 获取在 update 阶段已经更新过的 AaI(求逆结果)
54.         AaI_tmp = np.array([self.AaI[article] for article in articles])
55.         theta_tmp = np.array([self.theta[article] for article in articles])
56.         art_max = articles[np.argmax(np.dot(xaT, theta_tmp) + self.alpha * np.sqrt(np.dot(np.dot(xaT, Aa
    I_tmp), xa)))]
57.
58.         # 缓存选择结果，用于 update
59.         self.x = xa
60.         self.xT = xaT
```

```
        self.a_max = art_max

        return self.a_max
```

3.4. 尝试引入模拟退火

模拟退火（Simulated annealing，缩写作 SA）是一种通用概率算法，常用来在一定时间内寻找在一个很大搜寻空间中的近似最优解。模拟退火来自冶金学的专有名词退火。退火是将材料加热后再经特定速率冷却，目的是增大晶粒的体积，并且减少晶格中的缺陷。材料中的原子原来会停留在使内能有局部最小值的位置，加热使能量变大，原子会离开原来位置，而随机在其他位置中移动。退火冷却时速度较慢，使得原子有较多可能可以找到内能比原先更低的位置。

模拟退火的原理也和金属退火的原理近似：我们将热力学的理论套用到统计学上，将搜寻空间内每一点想像成空气内的分子；分子的能量，就是它本身的动能；而搜寻空间内的每一点，也像空气分子一样带有能量，以表示该点对命题的合适程度。算法先以搜寻空间内一个任意点作起始：每一步先选择一个邻居，然后再计算从现有位置到达邻居的概率。

加入模型的想法是基于一种对解决 EE（Exploit-Explore）问题的探索。我们可以在初期选择 arm 的时候加入模拟退火的思想，这样我们就不会每次都选择 reward 最大的臂并进行模型迭代。因为我们评判 reward 的累积遗憾是基于伯努利收益的设定，如果我们在迭代训练模型之后收敛的速度并没有得到很大的影响的话，我们在初期对于一个解决 EE（Exploit-Explore）问题的方案引入则显然是值得的。

```
# 就在这里改，改成前 K 的值，不在只选取最优解，看看效果
# max_p_t 变成一连串的值
# 再用模拟退火来约束
max_p_t = np.max(p_t)
post_energy = (-1)*(1/(1 + math.e ** (-10*(epoch+1))))
head_energy = (-1)*(1/(1 + math.e ** (-10*(epoch))))

energy_change = post_energy - head_energy
P = math.exp(energy_change)

top_k = 3
t_set = set(p_t)
t_list = list(t_set)
t_list.sort()
t_list.reverse()
sorted_t_list = t_list
top_k_p_t = sorted_t_list[:top_k]
tag = 0
while tag == 0:
    naive_p_t = np.random.choice(top_k_p_t)
    if max_p_t == naive_p_t:
        max_idxs = np.argwhere(p_t == naive_p_t).flatten()
        tag = 1
```

```
    else:
        pp = random.randint(0, 100)/100
        if pp >= P:
            max_idxs = np.argwhere(p_t == naive_p_t).flatten()
            tag = 1
        else:
            tag = 0
```

3.5. 尝试混合系数模型

在许多应用中，除了特定 arm 的功能外，使用所有 arm 共享的功能也很有帮助。例如，在新闻文章推荐中，用户可能只喜欢有关政治的文章，这就提供了一个机制。因此，拥有共享和非共享部分的特征是有帮助的。从形式上看，我们通过在前面的方程中加入另一个线性项来采用以下的混合模型。

$$E[r_{t,a}|x_{t,a}] = z_{t,a}^T\beta^* + x_{t,a}^T\theta_a^*$$

其中部分参数：

· $z_{t,a}^T$ = 当前实例组合的特征。

· β^* = 所有 arm 共有的未知系数向量。

基本算法描述与伪代码如下：

Algorithm 2 LinUCB with hybrid linear models.

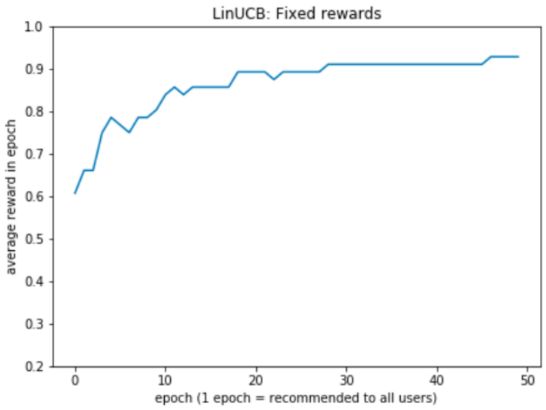
Inputs: $\alpha \in \mathbb{R}_+$
$\mathbf{A}_0 \leftarrow \mathbf{I}_k$ (k-dimensional identity matrix)
$\mathbf{b}_0 \leftarrow \mathbf{0}_k$ (k-dimensional zero vector)
for $t = 1, 2, 3, \ldots, T$ **do**
  Observe features of all arms $a \in \mathcal{A}_t$: $(\mathbf{z}_{t,a}, \mathbf{x}_{t,a}) \in \mathbb{R}^{k+d}$
  $\hat{\boldsymbol{\beta}} \leftarrow \mathbf{A}_0^{-1}\mathbf{b}_0$
  for all $a \in \mathcal{A}_t$ **do**
    if a is new **then**
      $\mathbf{A}_a \leftarrow \mathbf{I}_d$ (d-dimensional identity matrix)
      $\mathbf{B}_a \leftarrow \mathbf{0}_{d\times k}$ (d-by-k zero matrix)
      $\mathbf{b}_a \leftarrow \mathbf{0}_{d\times 1}$ (d-dimensional zero vector)
    end if
    $\hat{\boldsymbol{\theta}}_a \leftarrow \mathbf{A}_a^{-1}\left(\mathbf{b}_a - \mathbf{B}_a\hat{\boldsymbol{\beta}}\right)$
    $s_{t,a} \leftarrow \mathbf{z}_{t,a}^\top\mathbf{A}_0^{-1}\mathbf{z}_{t,a} - 2\mathbf{z}_{t,a}^\top\mathbf{A}_0^{-1}\mathbf{B}_a^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a} + \mathbf{x}_{t,a}^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a} + \mathbf{x}_{t,a}^\top\mathbf{A}_a^{-1}\mathbf{B}_a\mathbf{A}_0^{-1}\mathbf{B}_a^\top\mathbf{A}_a^{-1}\mathbf{x}_{t,a}$
    $p_{t,a} \leftarrow \mathbf{z}_{t,a}^\top\hat{\boldsymbol{\beta}} + \mathbf{x}_{t,a}^\top\hat{\boldsymbol{\theta}}_a + \alpha\sqrt{s_{t,a}}$
  end for
  Choose arm $a_t = \arg\max_{a\in\mathcal{A}_t} p_{t,a}$ with ties broken arbitrarily, and observe a real-valued payoff r_t
  $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
  $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
  $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{x}_{t,a_t}^\top$
  $\mathbf{B}_{a_t} \leftarrow \mathbf{B}_{a_t} + \mathbf{x}_{t,a_t}\mathbf{z}_{t,a_t}^\top$
  $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t\mathbf{x}_{t,a_t}$
  $\mathbf{A}_0 \leftarrow \mathbf{A}_0 + \mathbf{z}_{t,a_t}\mathbf{z}_{t,a_t}^\top - \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{B}_{a_t}$
  $\mathbf{b}_0 \leftarrow \mathbf{b}_0 + r_t\mathbf{z}_{t,a_t} - \mathbf{B}_{a_t}^\top\mathbf{A}_{a_t}^{-1}\mathbf{b}_{a_t}$
end for

4. 结果分析

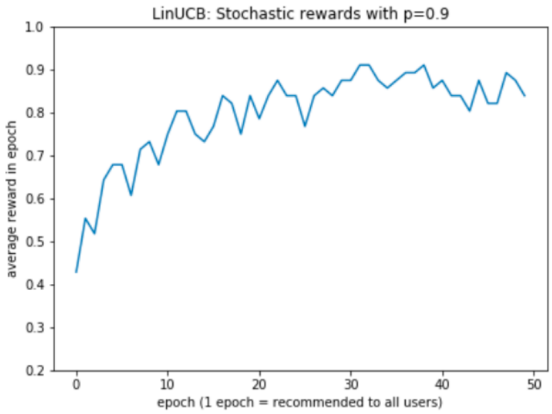
4.1. LinUCB

这是最经典的 LinUCB 算法实现，我们通过模型收敛的图像来分析模型的效率：

给定的 reward：

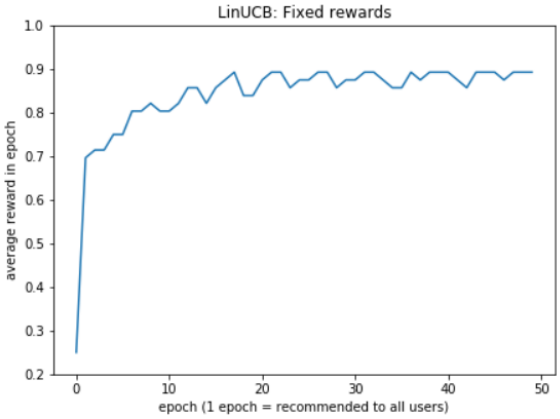


带随机性质的 reward：

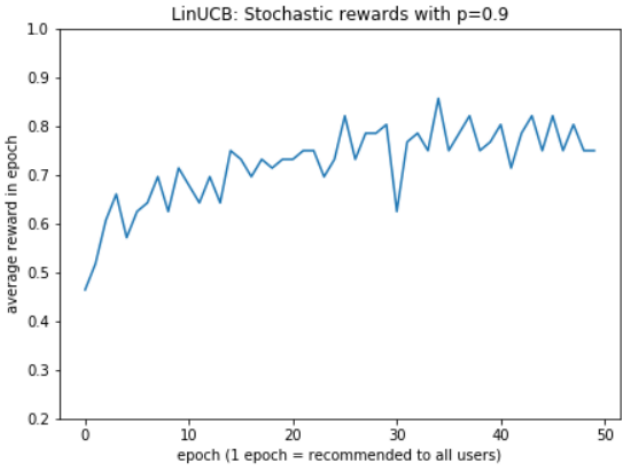


4.2. SaLinUCB

给定的 reward：

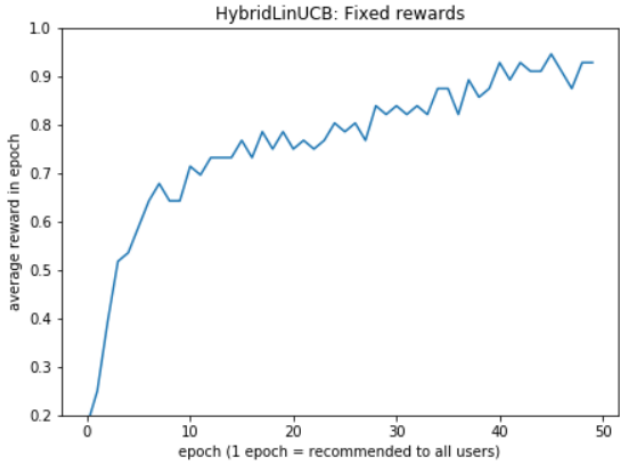


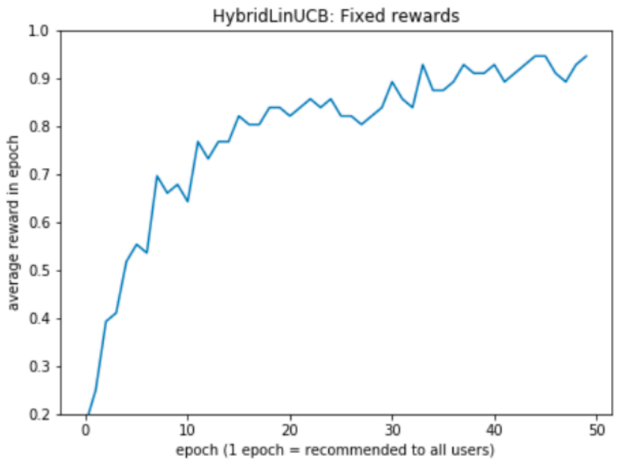
带随机性质的 reward：



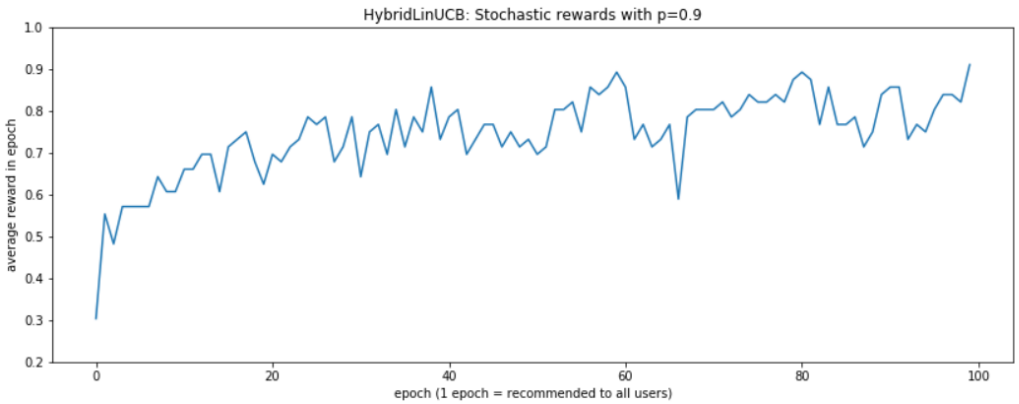
4.3. HybridUCB

给定的 reward：





带随机性质的 reward：



在进行模型改进后 SaLinUCB 与 HybridLinUCB 的收敛速度与遗憾在给定 reward 的情况下并没有很大削弱，反而我们相信改进的模型在解决 EE 与具有相同特征的实例推荐中具有更好的性能。这部分的具体性能分析本实验还没有深入探究，可以继续拓展。

5. 提交文件

data	#	著名电影推荐数据集 MovieLens 数据划分后的集合
_commons.py	#	异常处理
movielens.py	#	导入 movielens 数据并初始化
linUCB.py	#	naïve 的 linUCB 算法与加入模拟退火改进的 linUCB 算法
hybridLinUCB.py	#	加入共享系数的 linUCB 算法
LinUCB.ipynb	#	Jupyter Notebook 中运行 linUCB 中的算法
hybridLinUCB.ipynb	#	Jupyter Notebook 中运行 hybridLinUCB 中的算法

6. 引用

https://aws.amazon.com/cn/blogs/media/whats-new-in-recommender-systems/

Wikipedia contributors, "Multi-armed bandit," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Multi-armed_bandit&oldid=1026567376 (accessed June 21, 2021).

https://chenhh.gitbooks.io/multiperiod_portfolio_optimization/content/ml/bandit.html

Wikipedia contributors, "Simulated annealing," *Wikipedia, The Free Encyclopedia,* https://en.wikipedia.org/w/index.php?title=Simulated_annealing&oldid=1017509035 (accessed June 21, 2021).