推荐系统中的EE问题

# Exploit：利用 Explore：探索

推荐系统中，如果片面优化用户的喜好，推荐系统可能会造成信息茧房问题。也就是推荐的东西都是用户喜欢的东西，千篇一律。这时候不仅仅需要Exploit，还需要Explore新内容。另外呢，EE还可以通过Explore信息不足的物品（如很少曝光），提高对其信息的掌握程度（如ctr等）。随着时间的推移，推荐系统对item的信息掌握越来越多，也就可以更好的做出决策。

但是在工业界，EE算法其实是一个很矛盾的东西。上吧，确实可以提高新颖度，但是谁知道是正向影响还是负向影响。若不上EE，之前提到的种种好处也就无从谈起。但是，万一这些新东西很严重的损害了用户体验，造成用户流失，公司得哭死。所以，工业界对EE都有着很谨慎的态度。

EE算法这种思路是很好的，就是在实践中需要仔细斟酌使用方法，现在来看看工业界常用的EE算法：

首先是EG(Epsilon-Greedy)算法，这种算法思路很简单。以推荐场景为例，设定一个参数P，其中p%的人按照ctr大小顺序进行商品推荐，剩下的(1-p)%用户随机推荐商品。p%用户的策略为Exploit，(1-p)%用户的策略为Explore。这样造成的后果是这(1-p)%用户的体验可能会遭到很严重的损害，以至于流失。而另外p%的用户体验反而因为这(1-p)%的用户而变好了，因为推荐系统牺牲了他们的用户体验来换取商品的信息(ctr等)，让系统对商品的推荐更加准确，从而提升了p%用户按照ctr推荐的用户体验。EG算法的随机性太强了，会对用户体验造成很不好的影响，有没有什么办法可以减小这种影响呢？答案肯定是有的。比如筛选出商品的优选池，每次Explore的商品都在优选池中选。有的小伙伴可能会说既然规定优选池了，里面的商品都是经过人工筛选过了的，那么Explore还有意义吗？好，那我们在Explore时，一部分从优选池中选，另外一部分完全随机。如此，(1-p)%的用户即使有部分商品是随机呈现的，但因为有优选池商品的加持，体验损害有限。这些EG中减小用户体验损失的措施可以根据具体的业务场景再进行尝试。

第二个为UCB(upper confidence bound)算法。UCB算法是引入了置信度这个概念。这个置信度可以宽泛的去理解，大致逻辑是：该商品被选中的次数越多，其置信度就应该越高，越接近真实值。最终UCB是根据均值+置信度的和进行排序的。UCB算法舍弃了随机推荐的概念，其实就是一个确定性(deterministic)算法了，根据置信度产生的随机是伪随机。极端来说，若每个商品的均值就是真实值了，那么每个人看到的都是一样的。这可不就是确定性算法了嘛。

第三个为汤普森采样(Thompson Sampling )。汤普森采样先给商品的信息(ctr)定义了先验分布，然后利用每次的观察结果去计算后验分布，从每个商品的后验分布中采样生成随机数，取这些随机数中最大的进行推荐展示，依次循环。由于每一轮汤普森采样中，都有根据分布采样随机数的过程，所以汤普森采样是个随机的过程。为了每一轮迭代，方便先验概率与后验概率转化，可以使用共轭先验。即先验概率根据观察结果更新后验概率时，分布形式不变，只有参数发生变化。对于伯努利分布来说，共轭先验是Beta分布。在后续迭代中，根据实验结果更新Beta分布的a、b参数即可。这也就是为什么在ctr预估时贝叶斯平滑采用Beta先验分布的原因。

========================================

# 1探索与利用简介

在强化学习中，探索（Exploration ）的目的是找到更多有关环境的信息，而利用（Exploitation）的目的是利用已知的环境信息来最大限度地提高奖励。简而言之，探索是尝试还未尝试过的动作行为，而利用则是从已知动作中选择下一步的动作。

探索与利用之间的如何权衡，是强化学习的一个基本的问题。例如在很多情况，为了获得最佳的长期策略，可能需要做一些短期的牺牲。为了能够获得最佳的总体策略，往往需要收集到更多的信息。

有几种方式可以达到探索的目的：

* 第一种是朴素探索(Naive Exploration)，类似算法在一定概率基础下随机选择一些action。
* 第二种是乐观初始估计(Optimistic Initialization)，优先选择当前被认为是最高价值的行为，除非新信息的获取推翻了该行为具有最高价值这一认知；
* 第三种是不确定优先(Optimism in the Face of Uncertainty):，更加倾向选择更加具有不确定性的状态/动作，这种方法就需要一种方法来衡量这种不确定性
* 第四种是概率匹配（Probability Matching): 根据当前估计的概率分布采样行为；
* 第五种是信息状态搜索(Information State Search)，将已探索的信息作为状态的一部分联合个体的状态组成新的状态，以新状态为基础进行前向探索。

根据搜索过程中使用的数据结构，可以将搜索分为：

* 依据状态行为空间的探索(State-Action Exploration)，其针对每一个当前的状态，以一定的算法尝试之前该状态下没有尝试过的行为。
* 参数化搜索（Parameter Exploration)。直接针对策略的函数近似，此时策略用各种形式的参数表达，探索即表现为尝试不同的参数设置。

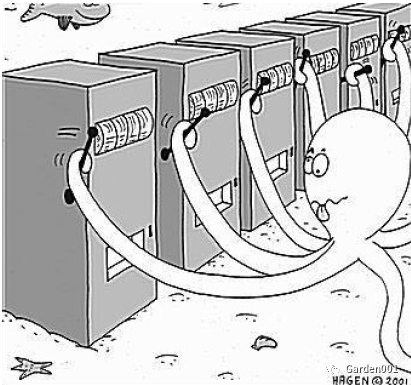
其优点是：得到基于某一策略的一段持续性的行为；

其缺点是：对个体曾经到过的状态空间毫无记忆，也就是个体也许会进入一个之前曾经进入过的状态而并不知道其曾到过该状态，不能利用已经到过这个状态这个信息。

为了较简单的描述各类搜索的原理，下一节将使用一种与状态无关的Bandit来进行讲解。

# 2、与状态无关的k-bandit问题

k-bandit问题考虑的是如下的学习问题：你要重复地在k个选项或者动作中进行选择。每次做出选择后，都会得到一定数值的收益，收益由你选择的动作决定的平稳概率分布产生。目标是在某一段时间内最大化总收益的期望。



该问题定义是：

multi-bandit可以看成是由行为空间和奖励组成的元组$，动作空间A是m个动作（即每个arms），每一个行为对应拉下某一个拉杆。奖励\mathcal{R}^{a}(r)=\mathbb{P}[r \mid a]$是未知的概率分布

在每个时间步t，智能体会选择动作，然后环境将生成奖励，目标为最大化累计收益。

# 3、后悔值（Regret）

为了方便描述问题，我们先给出几个定义：

## action-value

一个行为的价值等于该行为能得到的即时奖励期望，即该行为得到的所有即时奖励的平均值。



## optimal value

我们能够事先知道哪一个bandit能够给出最大即时奖励，那我们可以每次只选择对应的那个拉杆。如果用表示这个最优价值，表示能够带来最优价值的行为，则：



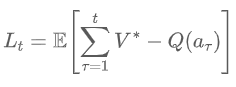
## Regret

事实上我们不可能事先知道拉下哪个拉杆能带来最高奖励，因此每一次拉杆获得的即时奖励可能都与最优价值存在一定的差距，我们定义这个差距为「后悔值(regret):」



## Total Regret

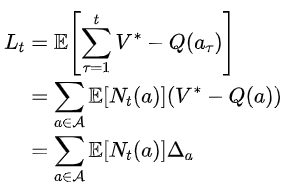
每做出一个行为，都会产生一个后悔值，因此随着持续的拉杆行为，将所有的后悔值加起来，形成「总后悔值(Total Regret)」：



这样，最大化累计奖励的问题就可以转化为最小化总后悔值了。之所以这样转换，是为了描述问题的方便，在随后的讲解中可以看到，较好的算法可以控制后悔值的增加速度。而用最大化累计奖励描述问题不够方便直观。

# 4、Regret的推导

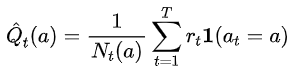
从另一个角度重写总后悔值。定义为到时刻t时已执行行为A的次数，定义差距（gap）为最优动作与行为a之间的差。那么总后悔值可以这样推导：



这相当于把个行为的差距与该行为发生的次数乘起来，随后把行为空间的所有行为的这个乘积再相加得到，只不过这里是期望。

把总后悔值用计数和差距描述可以使我们理解到一个好的算法应该尽量减少那些差距较大的行为的次数。不过我们并不知道这个差距具体是多少，因为根据定义虽然最优价值和每个行为的差距（gap）为静态的，但我们并不清楚这两者的具体数值，我们所能使用的信息就是每次行为带来的即时奖励r。那么我们如何利用每次行为的即时奖励呢？

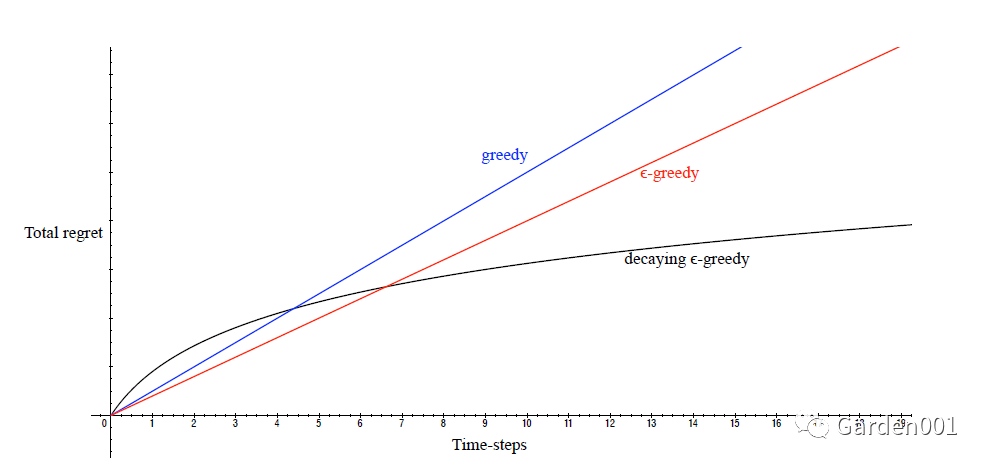
我们使用每次的即时奖励来计算得到t时刻止某一行为的平均价值：



这个方法也叫蒙特卡罗评估，以此来近似该行为的实际价值

# 5、Total Regret的直观理解

我们先直观了解下不同形式的随机策略其总后悔值随着时间的变化曲线：



（1）对于探索方法，总后悔值会呈线性增长，这是一个好的算法所不能接受的。这是因为每一个时间步，该探索方法有一定的几率选择最优行为，但同样也有一个固定小的几率采取完全随机的行为，如采取随机行为，那将一直会带来一定后悔值，如果持续以虽小但却固定的几率采取随机行为，那么总的后悔值会一直递增，导致呈现与时间之间的线性关系。类似的softmax探索方法与此类似。

（2）对于greedy探索方法，其总后悔值也是线性的，这是因为该探索方法的行为选择可能会锁死在一个不是最佳的行为上。

（3）目标就是找到一种探索方法，使用该探索方法时随着时间的推移其总后悔值增加得越来越少，后续将依次介绍几种较好的探索方法。

======================================================

# 背景

推荐系统通过对用户在APP里面的历史行为信息进行挖掘，向用户推荐与其历史行为相匹配的内容（商品、图文或者视频等），这其实就是一种对信息的Exploitation（利用）行为。但是推荐系统经常被诟病的问题是，总是推荐相似的信息，缺乏新意，容易造成审美疲劳，这也是很多用户在使用相关APP一段时间后选择离开的重要原因。如果系统里面存在用户感兴趣的内容，但是却没有让用户方便地获取到相应的信息，说明当前的推荐系统有问题。用户越看什么系统越推什么，慢慢造成推荐的信息形式越来越窄，从而形成信息茧房，这是当前推荐系统普遍会遇到的问题。

破圈的方法之一是Exploration（探索），即通过一些方法去探索用户可能的兴趣空间，不断扩大用户的兴趣边界，甚至发掘用户自己都不曾意识到的新的兴趣点，如果推荐系统能够做到这一点，那不夸张的说，对整个社会的进步都会有不小的促进作用。不断扩大用户认知边界，是为开放，收缩用户既有的认知，是为封闭。当然，有些人知道自己需要什么，会主动的去选择信息，但是在信息爆炸的时代，绝大多数人都是在被动的吸收信息，此时信息的分发方式影响就很大了，一个好的推荐系统，应该是能够不断地扩大用户的认知边界，而不是将用户包裹在信息茧房里，这应该作为做推荐系统的初心。

## Exploration方法

常见的Exploration方法有，朴素Bandit、Epsilon-Greedy、UCB、Thompson Sampling,、 LinUCB、COFIBA等，但是这些方法在当前的推荐系统中其实用得很少，主要原因是Exploration方法往往有瞎猜的性质，因为不能再完全根据既往的信息做决策。Exploration的意思就是在不断的试探当中拓展用户的兴趣边界，但是试探是有代价的，如果推出来的东西用户一点兴趣没有，久而久之，用户就失望了，从而选择离开，这样的推荐系统就更没有价值了，连最起码的商业目的都没有达到。所以，做Exploration相关的尝试，往往是针对老用户、死忠粉，同时，选择的内容池子质量也更高，至少做到推出去的内容用户不喜欢，但也不能让人讨厌。

* 朴素Bandit根据历史信息选择平均回报最高的，是一种贪心算法。
* Epsilon-Greedy在朴素Bandit的基础上增强了探索能力，以ε的概率做随机选择。
* UCB算法在朴素Bandit的基础上根据选择总次数及每个Item被选择的次数计算一个置信区间，然后根据平均分及置信分的和大小做选择。
* Thompson Sampling根据每个Item的beta分布产生的随机数去做选择，同时根据选择结果更新beta分布的参数。
* LinUCB是引入特征与监督学习的UCB算法。
* COFIBA是协同过滤结合Bandit的算法，User-based协同过滤来选择要推荐的Item，选择时使用了LinUCB思想，同时根据用户反馈更新相关矩阵参数。

以上方法，就类Epsilon-Greedy的方法在强化学习推荐系统当中还会使用。对于大规模离散特征的深度学习推荐系统，值得一试的Exploration策略有：

* 扰动训练好的深度模型NN参数，具体操作为对NN参数加上一定的高斯噪声。
* 随机丢弃部分强记忆型的ID类特征，如UserId和DocId。

策略一正常训练和保存NN参数，在Serving的时候向NN参数加入高斯噪声。高斯噪声为标准分布的随机数，通过当中的来控制噪声的强度。我们尝试过几组参数，0.1，0.01，0.001，0.0001,发现0.001能够取得比较好的探索和利用的平衡。

策略二在预估的时候，随机将部分id类特征的embedding置0，相当于不利用已经学习好的这些slot的embedding。因为像uid，did这类特征一般起到的就是记忆的作用，如果记忆能力太强了，那就容易影响到泛化能力。通过随机将部分id类特征的embedding置0，可以提升模型泛化能力，同时提升模型的探索能力。

这两个策略基本都达到了预期目的，用户的阅读散度，文章的多样性都能一定程度的提升，同时大盘实时指标并没有下降，甚至还有一定程度的提升，长留也是正向的。

# 其它思考

在信息流产品中，用户大部分的PV和时长都贡献给了主推荐页，可以通过子频道页增强对用户的探索能力。在特征构建的时候，我们谈到可以通过增加相似用户阅读文章和相似文章聚类特征来增强对新用户和新文章的探索能力。但是现实是，其它子频道页被大部分用户点开的机会很少，增加探索特征，但是模型仍然倾向于去记忆历史。

可以尝试使用个别探索性质强的召回做一下强插，比如说User-CF，因为精排就是一个贪心算法，即使召回出来了不错的内容也有可能被精排给干掉，使用强探索能力的召回做推送或者强插，结合精选内容池，说不定能够取得不错效果呢。

然后就是强化学习，以定义的长期收益作为目标，在与用户的交互过程当中持续进行学习。

提升探索能力往往意味着用户体验的降低甚至流失，但是持续给用户推荐相同类型的内容，用户也会逐渐阅读疲劳从而流失。好的推荐系统在于在Exploration & Exploitation之间取得一个比较好的平衡，在不严重影响用户阅读体验的前提下，时不时地给用户推荐一些好玩的、新奇的或者新颖的东西，探索甚至放大用户的兴趣，这也是做推荐系统的乐趣所在。

================================================

无论什么样的推荐系统总有两个绕不开的经典问题：EE(Exploration and Exploitation)和冷启动。

大家都知道推荐系统的本质是挖掘、学习并迎合用户兴趣，以此追求自己的利益最大化，那么怎么样的推荐系统才会带来利益最大化呢？

大家思考一个问题，假如你是一个码农，每天在今日头条上浏览技术相关的问题，这是你当下的兴趣，显然头条的推荐系统会学习到你的兴趣并持续不断的给你推荐技术相关的文章，但是，如果你每次一打开头条推荐的都是技术相关的文章，是不是很反感，喜欢技术但我也有其他的兴趣对吧，比如这个这个时候如果推送一个梅西要离开巴萨的新闻我可能更有兴趣，有的时间看技术文章看多了我也会有思想疲劳的时候，想看看其他的放松一下，这个时候如果来一个LGD淘汰IG进入S10世界赛，我也会更开心。这个问题的背后就是推荐中比较关注的两个指标准确性和多样性，一个好的推荐系统就是在准确性和多样性之间做tradeoff。也是推荐系统中的Exploration and Exploitation(EE问题，探索与开发)问题。

在EE问题中，Exploitation表示对用户比较确定的兴趣，就像上面问题中喜欢浏览技术文章，当然需要迎合，这代表当下的利益；而exploration表示只对着用户已知的兴趣使用，用户很快会腻，兴趣会逐渐降低，所以要不断探索用户新的兴趣，就像上面问题中的梅西转会和LGB进入S10，这代表未来的利益。

那么怎么样去解决的是推荐系统中EE问题呢？

Bandit算法是解决EE问题的一种有效算法，有一个很经典的问题，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呢？这就是多臂赌博机问题（Multi-armed bandit problem, K-armed bandit problem, MAB）

下面一起来讨论一下解决推荐系统中EE问题的一些经典Bandit算法。

# 朴素Bandit算法

先随机试若干次，计算每个臂的平均收益，每次迭代的时选收益均值最大那个臂，显然这样的方式比随机选择好。

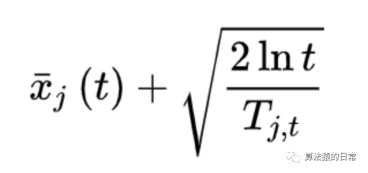
# Epsilon-Greedy算法

随机选一个(0,1)之间较小的数Epsilon，每次以Epsilon的概率在所有臂中随机选一个。以1-Epsilon的概率选择截止当前，平均收益最大的那个臂。根据选择臂的回报值来对回报期望进行更新。这里Epsilon的值可以控制对Exploitation和 Exploration的偏好程度，每次决策以概率Epsilon去 Exploration，1-Epsilon的概率来Exploitation。

对于Epsilon-Greedy算法来说，能够应对变化，能够根据收益及时改变策略。同时Epsilon的值可以控制对Exploration和Exploitation的偏好程度。Epsilon越接近0，说明只在乎眼前的苟且，不看重远方的田野。但是推荐策略运行一段时间后，我们已经积累了一定的知识，但却没有好好利用，仍然做随机Exploration，显然这是不可取的，这是Epsilon-Greedy算法的缺点。

# UCB(Upper Confidence Bound, 置信区间上界)算法

UCB 算法全称是 Upper Confidence Bound（置信区间上界）。先对每一个臂都试一遍，之后在任意时刻t(t>K)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



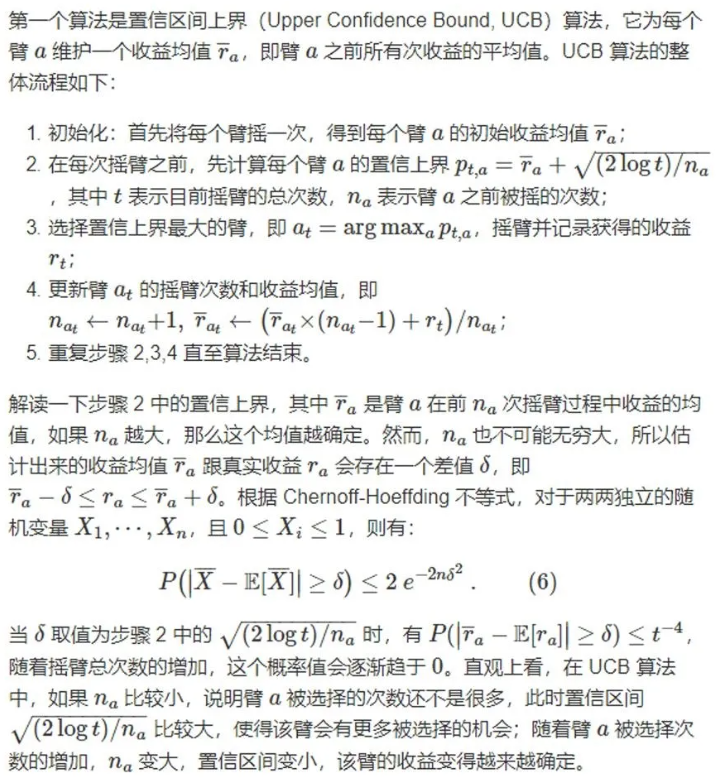
观察选择结果，更新图片和Tj,t。其中表示的是第j个老虎机到目前的平均收益，，后面的叫做bonus，本质上是均值的标准差，图片是目前的试验总次数，Tj,t是该臂试验次数。

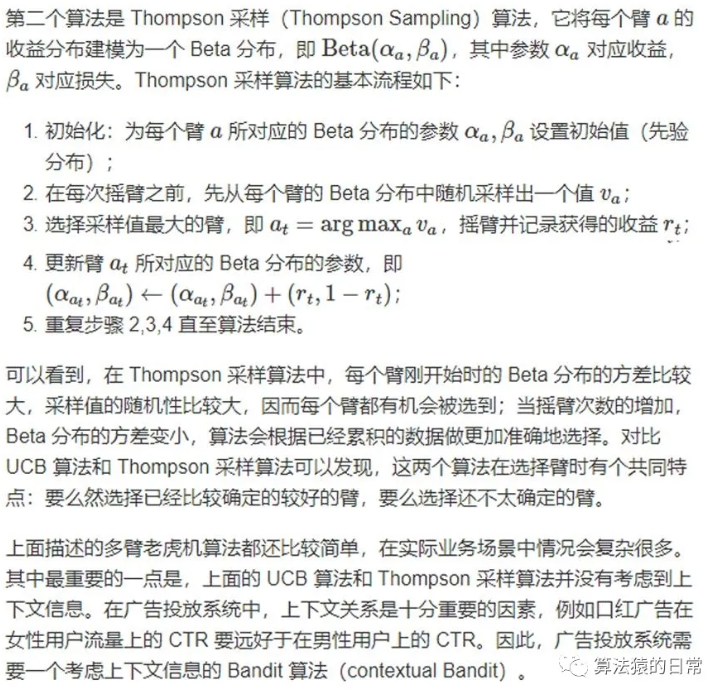
|  |
| --- |
| # 计算delta  # N：老虎机的个数  # chosen\_count: N维向量，表示每个老虎机被选中的次数  # item：当前老虎机  delta = np.sqrt(2 \* np.log(T) / chosen\_count[item])  # 计算每个老虎机的p+delta，选择最优  # t: 第t次实验  # estimated\_rewards:item老虎机平均收益  # true\_rewards：真实收益  item = np.argmax([estimated\_rewards[item] + calculate\_delta(t, item) for item in range(N)])  reward = np.random.binomial(n=1, p=true\_rewards[item])  # 更新平均收益  estimated\_rewards[item] = ((t - 1) \* estimated\_rewards[item] + reward) / t |

# Thompson sampling算法

Thompson sampling算法用数据分布的思想去思考这个问题，该方法假设每个老虎机都对应一个吐钱概率p，同时概率p的概率分布符合beta(分布，每个臂都维护一个beta分布的参数，它有两个参数wins, lose。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。

|  |
| --- |
| # 维护一个beta分布，获取最大的随机数  choice = np.argmax(pymc.rbeta(1 + wins, 1 + trials - wins)) |



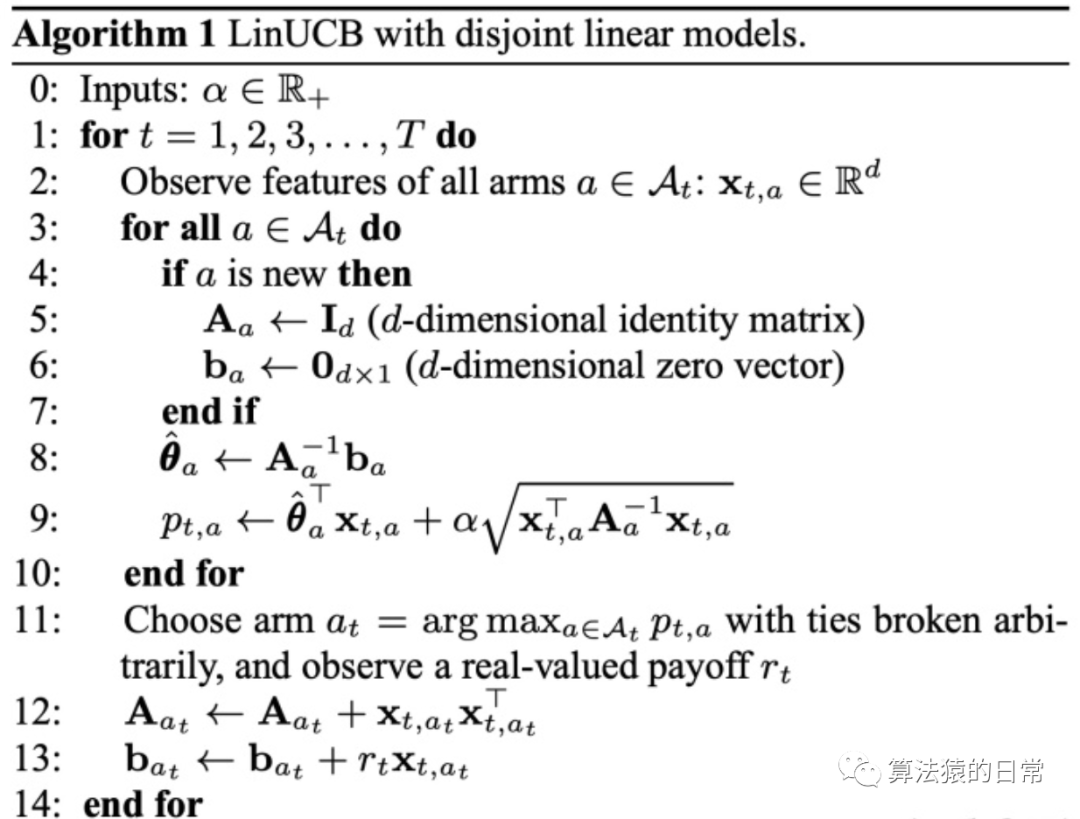


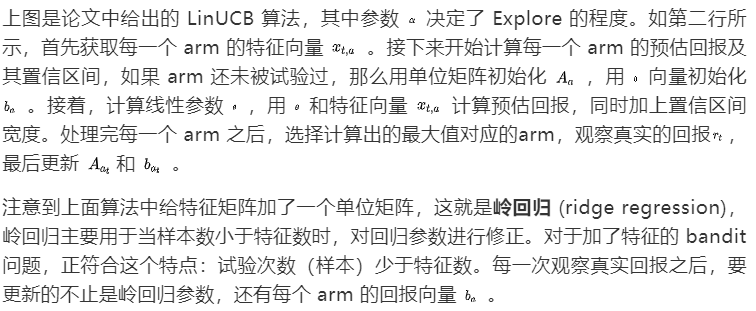
# LinUCB算法

UCB 算法虽然做很好的解决了EE ( Exploit-Explore ) 问题，但有它有一个缺点即是没有考虑上下文信息。

Yahoo在2010年发表了一篇论文提出了一种改进的UCB算法，称之为LinUCB，在Yahoo的新闻推荐表现更突出，就是在UCB基础上引入。单纯的老虎机收益情况就是老虎机自己内部决定的，而在推荐领域，一个选择的回报，是由User和Item一起决定的，如果我们能用context来刻画User和Item的关系，在每次选择Item之前，通过context预估每一个arm的期望收益及置信区间，选择的收益就可以通过context泛化到不同的arm上。

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，以此达到试验学习的目的。





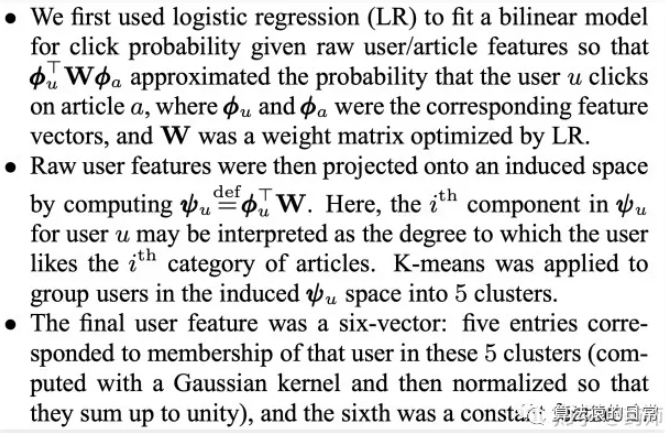
LinUCB 算法有一个很重要的步骤，就是给User和Item构建特征，也就是刻画context。在原始论文里，Item是文章，下面是构建的特征。

**原始用户特征:**

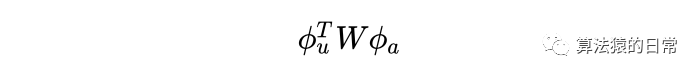
* 人口统计学，性别特征（2 类），年龄特征（离散成 10 个区间）；
* 地域信息，遍布全球的大都市，美国各个州；
* 行为类别，代表用户历史行为的 1000 个类别取值。

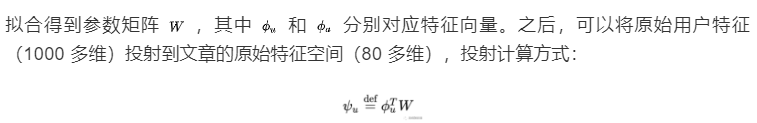
**原始文章特征：**

* URL类别，根据文章来源分成了几十个类别；
* 编辑打标签，编辑人工给内容从几十个话题标签中挑选出来的。



原始特征向量都要归一化成单位向量。还要对原始特征降维，以及模型要能刻画一些非线性的关系。用 Logistic Regression 去拟合用户u对文章a的点击历史，其中的线性回归部分为：





然后，用投射后的 80 多维特征对用户聚类（可以使用 K-means），得到 5 个类簇，文章页同样聚类成5个簇，再加上常数1，用户和文章各自被表示成6维向量。Yahoo! 的科学家们之所以选定为 6 维，因为数据表明它的效果最好，并且这大大降低了计算复杂度和存储空间。

LinUCB 算法有以下优点：由于加入了特征，所以收敛比 UCB 更快（论文有证明）；特征构建是效果的关键，也是工程上最麻烦和值得发挥的地方；由于参与计算的是特征，所以可以处理动态的推荐候选池，编辑可以增删文章；特征降维很有必要，关系到计算效率。

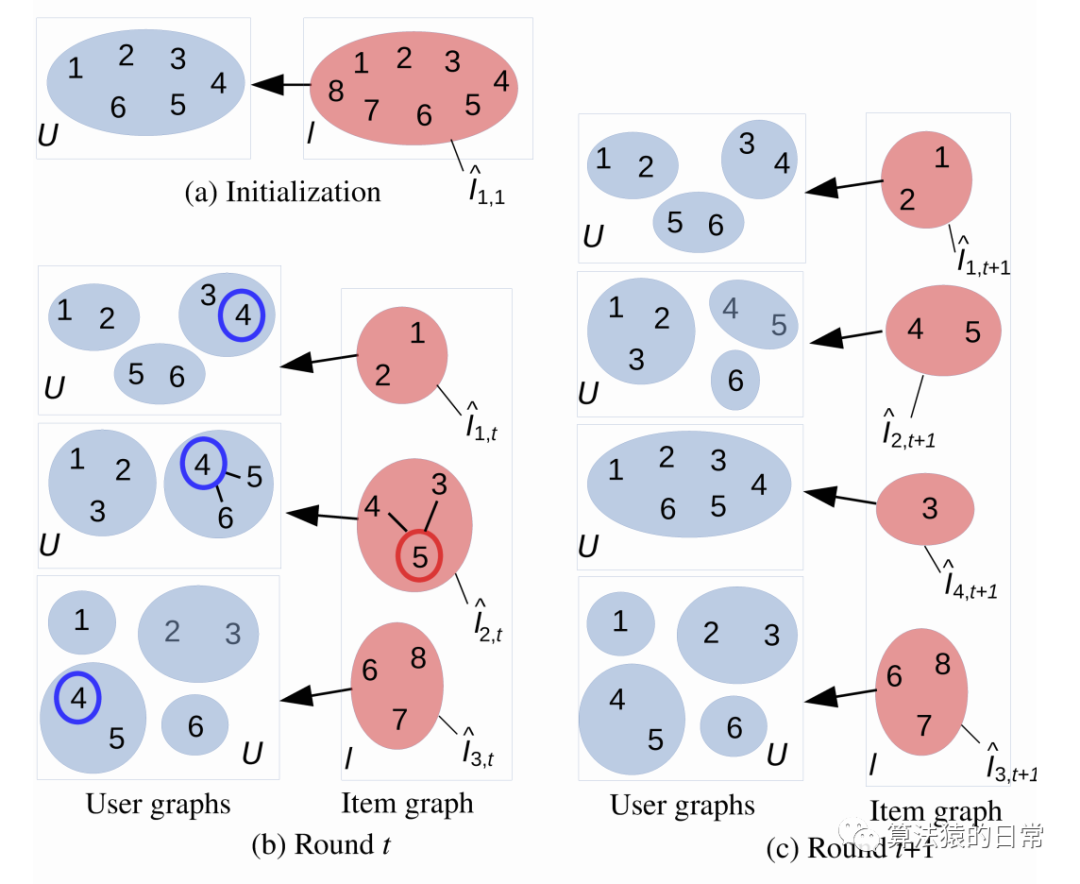
# COFIBA算法

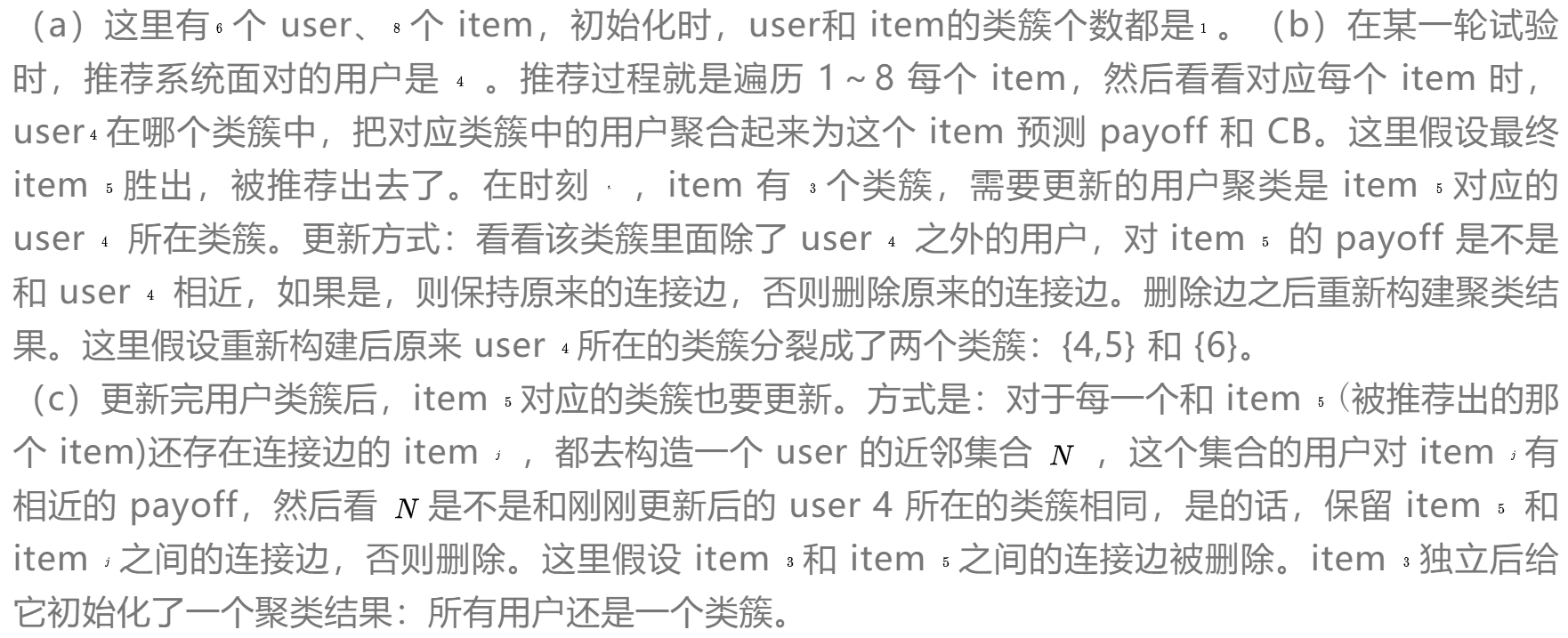
推荐系统中最经典的算法莫过于协同过滤了，它的核心假设就是："物以类聚，人以群分"，而解决EE问题的核心是：基于收益最大化寻找长短期兴趣的tradeoff，bandit算法是解决EE问题非常好的方式，而在上一节中LinUCB提出了结合UCB和上下文解决EE问题，显然其根本思想在于user&item上下文信息是对Bandit补充。在这个理念下，基于Bandit算法与协同过滤结合的方法COFIBA被提了出来。

COFIBA算法的思想是在时刻 t，用户来访问推荐系统，推荐系统需要从已有的候选池子中挑一个最佳的物品推荐给他，然后观察他的反馈，用观察到的反馈来更新挑选策略。这里的每个物品都有一个特征向量，所以这里的 bandit 算法是 context 相关的。这里依然是用岭回归去拟合用户的权重向量，用于预测用户对每个物品的可能反馈 (payoff)，这一点和 linUCB 算法是一样的。对比 LinUCB 算法，COFIBA 算法的不同有两个：基于用户聚类挑选最佳的 item（相似用户集体决策的 bandit）；基于用户的反馈情况调整 user 和 item 的聚类（协同过滤部分）。

算法针对某个用户 图片 ，首先计算该用户的 bandit 参数 w(和 LinUCB 相同)，但是这个参数并不直接参与到 bandit 的选择决策中(和 LinUCB 不同)，而是用来更新用户聚类的；接着遍历候选 item，每一个 item 表示成一个 context 向量；这样，每一个 item 都对应一套用户聚类结果，所以遍历到每一个 item 时判断当前用户在当前 item 下属于哪个类簇，然后把对应类簇中每个用户的M矩阵 (对应 LinUCB 里面的 A 矩阵)，b向量（payoff 向量，对应 LinUCB 里面的b向量）聚合起来，从而针对这个类簇求解一个岭回归参数(类似 LinUCB 里面单独针对每个用户所做)，同时计算其 payoff 预测值和置信上边界；然后，每个 item 都得到一个 payoff 预测值及置信区间上界，挑出那个上边界最大的 item 推出去(和 LinUCB 相同)；观察用户的真实反馈，然后更新用户自己的M矩阵和b向量(更新个人的，对应类簇里其他的不更新)。

以上是 COFIBA 算法的一次决策过程。在收到用户真实反馈之后，还有两个计算过程：更新 user 聚类和更新 item 聚类。论文中给出的更新 user 和 item 的聚类示意图如下所示。





（a）这里有图片个 user、图片个 item，初始化时，user和 item的类簇个数都是图片。（b）在某一轮试验时，推荐系统面对的用户是 图片 。推荐过程就是遍历 1～8 每个 item，然后看看对应每个 item 时，user图片在哪个类簇中，把对应类簇中的用户聚合起来为这个 item 预测 payoff 和 CB。这里假设最终 item 图片胜出，被推荐出去了。在时刻 图片 ，item 有 图片个类簇，需要更新的用户聚类是 item 图片对应的user 图片 所在类簇。更新方式：看看该类簇里面除了 user 图片 之外的用户，对 item 图片 的 payoff 是不是和 user 图片 相近，如果是，则保持原来的连接边，否则删除原来的连接边。删除边之后重新构建聚类结果。这里假设重新构建后原来 user 图片所在的类簇分裂成了两个类簇：{4,5} 和 {6}。

（c）更新完用户类簇后，item 图片对应的类簇也要更新。方式是：对于每一个和 item 图片(被推荐出的那个 item)还存在连接边的 item 图片 ，都去构造一个 user 的近邻集合 图片 ，这个集合的用户对 item 图片有相近的 payoff，然后看 图片是不是和刚刚更新后的 user 4 所在的类簇相同，是的话，保留 item 图片 和 item 图片之间的连接边，否则删除。这里假设 item 图片和 item 图片之间的连接边被删除。item 图片独立后给它初始化了一个聚类结果：所有用户还是一个类簇。

COFIBA算法有如下几个特点：

* User-based协同过滤来选择要推荐的item，选择时用了LinUCB的思想
* 根据用户的反馈，调整User-based和Item-based的聚类结果
* Item-based的聚类变化又改变了User的聚类
* 不断根据用户实时动态的反馈来划分User-Item 矩阵

# 参考

<https://mp.weixin.qq.com/s/FkDorCrlBcR_bkVFBw0ruA>

<https://mp.weixin.qq.com/s/VpylJMz0xdmobzPNiAkP6g>

<https://mp.weixin.qq.com/s/N3n7aegr6wYIhCF7yeSSSg>

<https://mp.weixin.qq.com/s/CWFbBLaRhBt0iWOL9VnPfg>