推荐系统中的EE问题

# 1 推荐系统中的EE问题

Exploration and Exploitation（EE问题，探索与开发）是计算广告和推荐系统里常见的一个问题。为什么会有EE问题？这是因为推荐系统通过对用户历史行为进行挖掘，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），这其实是一种对信息的利用（Expoitation）行为。但如果总是推荐相似的信息，缺乏新意，容易造成审美疲劳，这也是有些用户在使用相关app一段时间后选择离开的原因之一。如果用户越看什么就越推荐什么，从一方面来讲确实是准确率很高，但是这种会形成信息茧房。因此，在推荐系统中，不仅需要充分利用用户的历史信息，还需要探索用户新的喜好，这就是探索和利用的思想。探索与利用就是为了平衡推荐系统的准确性和多样性，在进行物品推荐时，不仅要投其所好，还要进行适当的长尾物品挖掘。

* Exploitation：对于用户已经确定的兴趣当然要迎合投放；
* Exploration：除了推荐已知的用户感兴趣的内容，还需要不断探索用户其他兴趣。

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，可能造成用户流失。所以，工业界对EE都有着很谨慎的态度。因此，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。

# 2 MAB 问题

问题来源于历史悠久的赌博学，它要解决的问题是这样的：

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）。

怎么解决这个问题呢？最好的办法是去试一试，不是盲目地试，而是有策略地快速试一试，这些策略就是Bandit算法。

Bandit算法如何同推荐系统中的EE问题联系起来呢？假设我们已经经过一些试验，得到了当前每个老虎机的吐钱的概率，如果想要获得最大的收益，我们会一直摇哪个吐钱概率最高的老虎机，这就是Exploitation。但是，当前获得的信息并不是老虎机吐钱的真实概率，可能还有更好的老虎机吐钱概率更高，因此还需要进一步探索，这就是Exploration问题。

这个经典问题集中体现了推荐系统中一个核心的权衡问题：我们是应该探索（exploration），去尝试挖掘用户新的兴趣，还是应该守成（exploitation），坚持目前已知的用户兴趣？在多臂老虎机问题中，探索意味着去玩还没玩过的老虎机，但这有可能使你花太多时间和金钱在收益不好的机器上；而守成意味着只玩目前为止给你收益最好的机器，但这又可能使你失去找到更好机器的机会。

# 3 Bandit算法

常见的Exploration（探索）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。

Bandit算法是在线学习的一种，一切通过数据收集进行的概率预估任务，都可以通过Bandit系列算法来进行在线优化。这里的“在线”，并不是指互联网的线上，而是指算法模型参数根据观察数据不断变化。

如何将Bandit算法、MAB问题、推荐系统中的EE问题三者联系起来呢？

假设我们已经经过一些试验，得到了当前每个老虎机的吐钱的概率预估值，如果想要获得最大的收益，我们会一直摇哪个吐钱概率预估值最高的老虎机，这就是Exploitation。但是，当前获得的信息并不是老虎机吐钱的真实概率，可能还有吐钱概率更高的老虎机没有被我们试验出来，因此还需要进一步探索，这就是Exploration问题。

Bandit算法中有几个关键元素：臂，回报，环境。

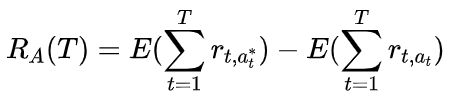
* 臂：指每次选择的候选项，有几个选项就有几个臂。
* 回报：就是选择一个臂之后得到的奖励，比如选择赌博机后吐出来的硬币。
* 环境：就是决定每个臂不同的那些因素，统称为环境。

将以上的关键元素对应到推荐系统中。

* 臂：指每次推荐的候选项，可以是具体物品，也可以是物品类别，也可以是推荐策略和算法。
* 回报：指用户对推荐的结果是否满意。
* 环境：指给当前用户推荐时的所有周边环境

如何衡量Bandit算法的优劣？

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？能不能遗憾少一些？所以我们便有了衡量Bandit算法的一个指标：累积遗憾。



这里t表示当前选择的轮数；T表示总共选择的轮数；表示经过T次选择后的累计遗憾；表示在第t次选择时选择了最好的臂所获得的收益；表示在第t次选择时实际所选的臂所带来的收益。公式右边的第一项表示第t轮的期望最大收益，而右边的第二项表示第t轮实际选择的臂获取的收益，把每次差距累加起来就是总的遗憾。

Bandit算法的套路就是：小心翼翼地试，越确定某个选择好，就多选择它，越确定某个选择差，就越来越少选择它。如果某个选择实验次数较少，导致不确定好坏，那么就多给一些被选择机会，直到确定了它是金子还是石头。简单说就是，把选择的机会给“确定好的”和“还不确定的”。

有了衡量算法优劣的指标就来看一看具体的Bandit算法吧。

对应同样的问题，采用不同bandit算法来进行实验相同的次数，那么看哪个算法的总regret增长最慢，那么哪个算法的效果就是比较好的。

## 3.1 朴素Bandit算法

核心思想：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。

这个算法是我们普通人实际生活中最常采用的，不可否认，它还是比随机乱猜要好。属于贪心算法。

## 3.2 Epsilon-Greedy算法

先随机选一个(0,1)之间较小的数epsilon，每轮以概率epsilon在所有臂中随机选一个臂，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。

简单清晰地以epsilon的值控制对exploit和explore的偏好程度，每次决策以概率epsilon去勘探（Exploration），以1-epsilon概率来开发（Exploitation），基于选择的item及回报，更新item的回报期望。

优点：能够应对变化，即如果item的回报发生变化，能及时改变策略，避免卡在次优状态。同时的值可以控制对Exploit和Explore的偏好程度。

缺点：策略运行一段时间后，我们已经对各item有了一定程度了解，但没有充分利用这些信息，仍然不做任何区分地随机Exploration，这是Epsilon-Greedy算法的缺点。

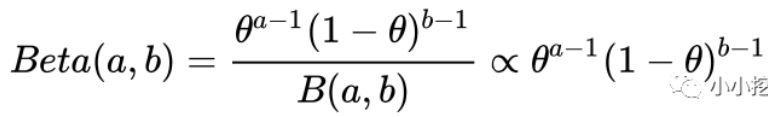
采用该算法造成的后果是p%的用户的体验可能会遭到很严重的损害，以至于流失。而另外(1-p)%的用户体验反而因为这p%的用户而变好了，因为推荐系统牺牲了他们的用户体验来换取商品的信息(ctr等)，让系统对商品的推荐更加准确，从而提升了(1-p)%用户按照ctr推荐的用户体验。EG算法的随机性太强了，会对用户体验造成很不好的影响，有没有什么办法可以减小这种影响呢？答案肯定是有的。比如筛选出商品的优选池，每次Explore的商品都在优选池中选。有的小伙伴可能会说既然规定优选池了，里面的商品都是经过人工筛选过了的，那么Explore还有意义吗？好，那我们在Explore时，一部分从优选池中选，另外一部分完全随机。如此，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。

## 3.3 汤普森采样（Thompson sampling）算法

该方法基于Beta分布，假设每个老虎机都有一个吐钱的概率p，同时该概率p的概率分布符合Beta(wins,lose)分布，每个臂都维护一个Beta分布的参数，即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。

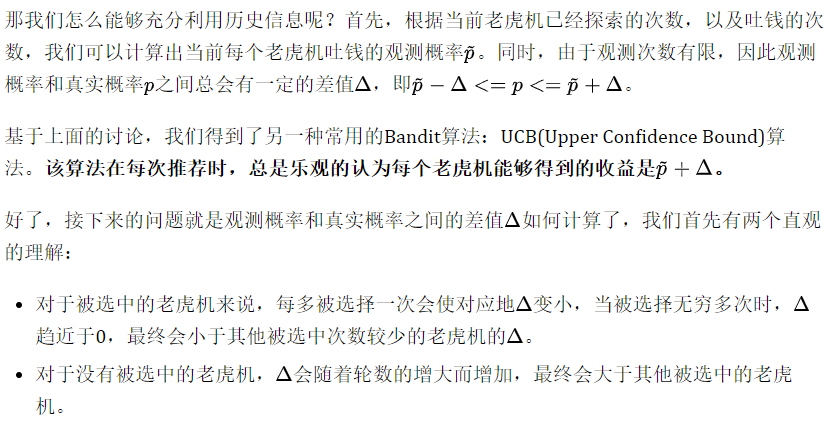
beta分布可以看作一个概率的概率分布。它是对二项分布中成功概率p的概率分布的描述。它的形式如下：

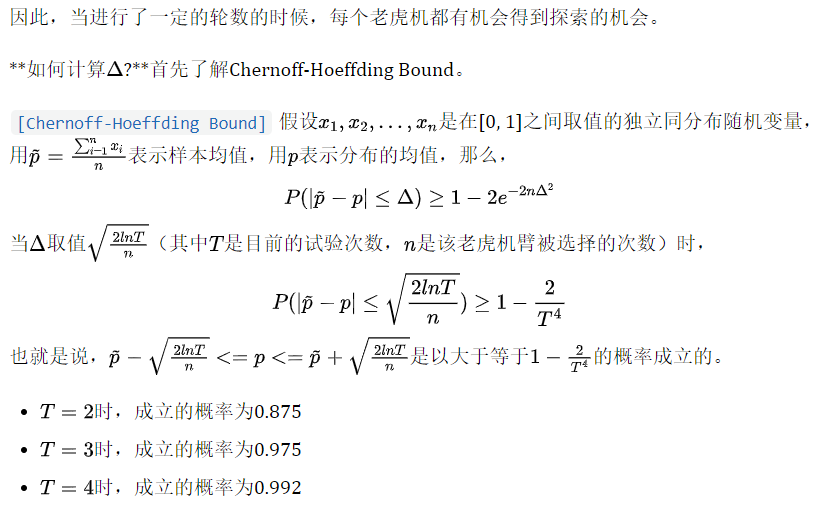


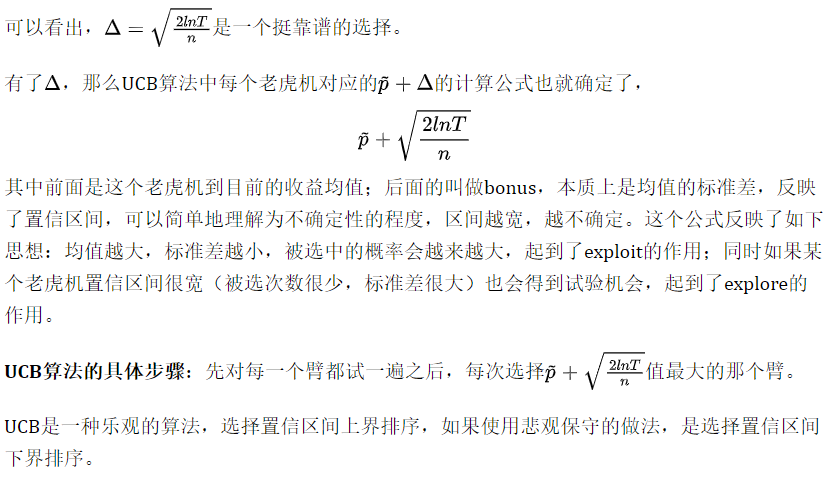
其中，a和b分别代表在a+b次伯努利试验中成功和失败的次数。

## 3.4 UCB算法

前面提到了，Epsilon-Greedy算法在探索的时候，所有的老虎机都有同样的概率被选中，这其实没有充分利用历史信息，比如每个老虎机之前探索的次数，每个老虎机之前的探索中吐钱的频率。





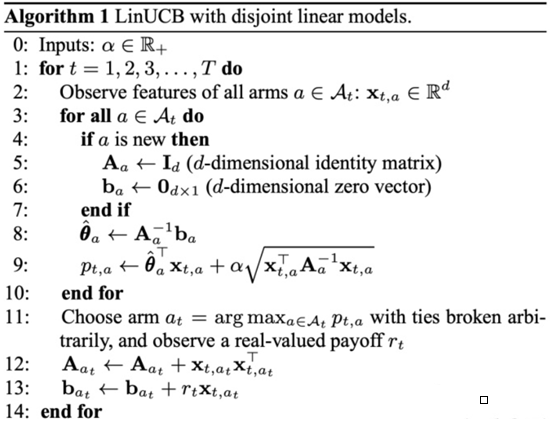


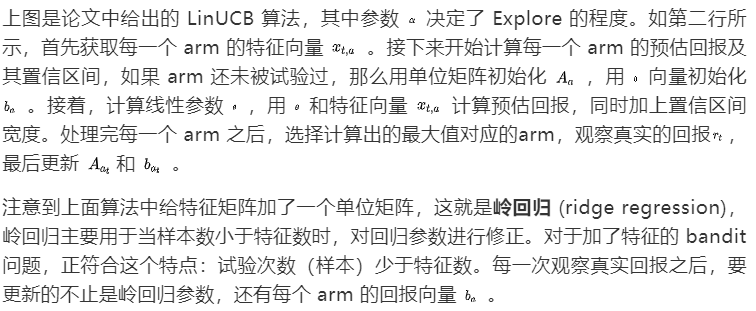
## 3.5 LinUCB算法

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：**

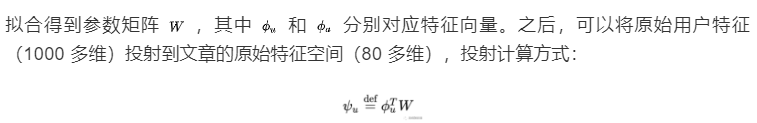
* 人口统计学，性别特征（2 类），年龄特征（离散成10个区间）；
* 地域信息，遍布全球的大都市，美国各个州；
* 行为类别，代表用户历史行为的1000个类别取值。

**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。

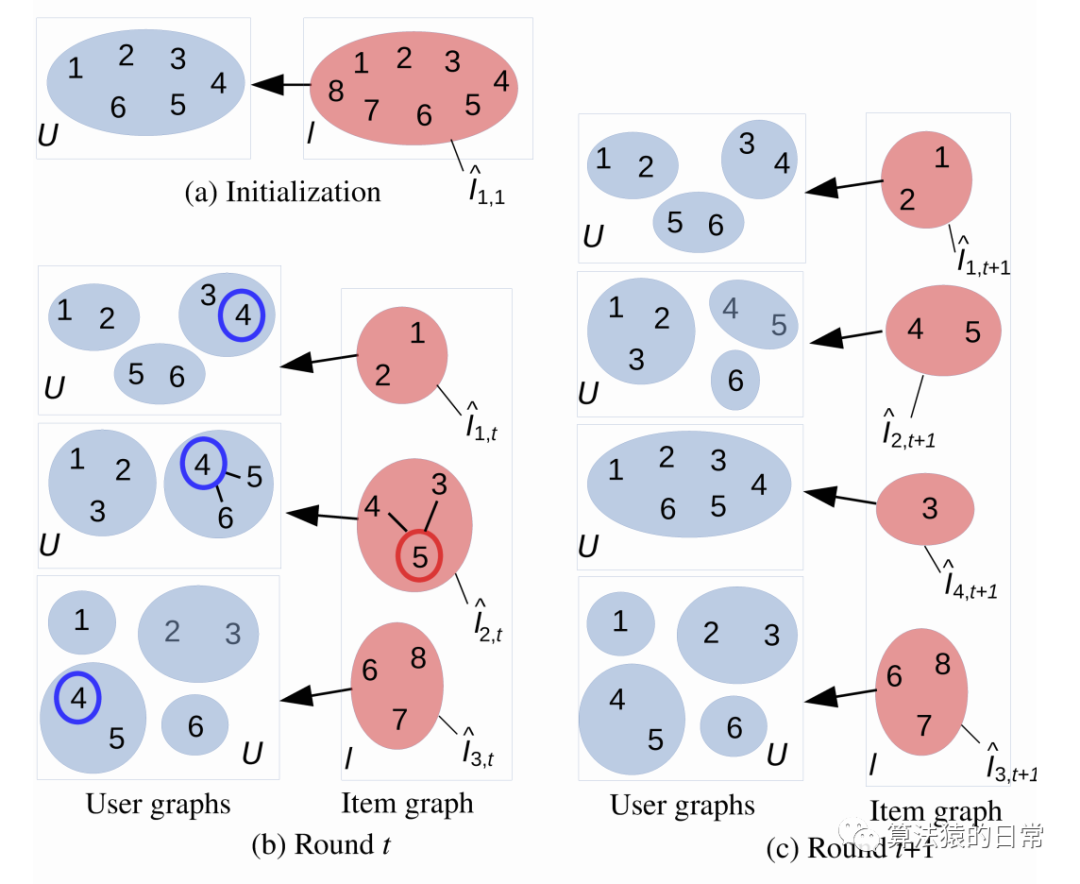
## 3.6 COFIBA算法

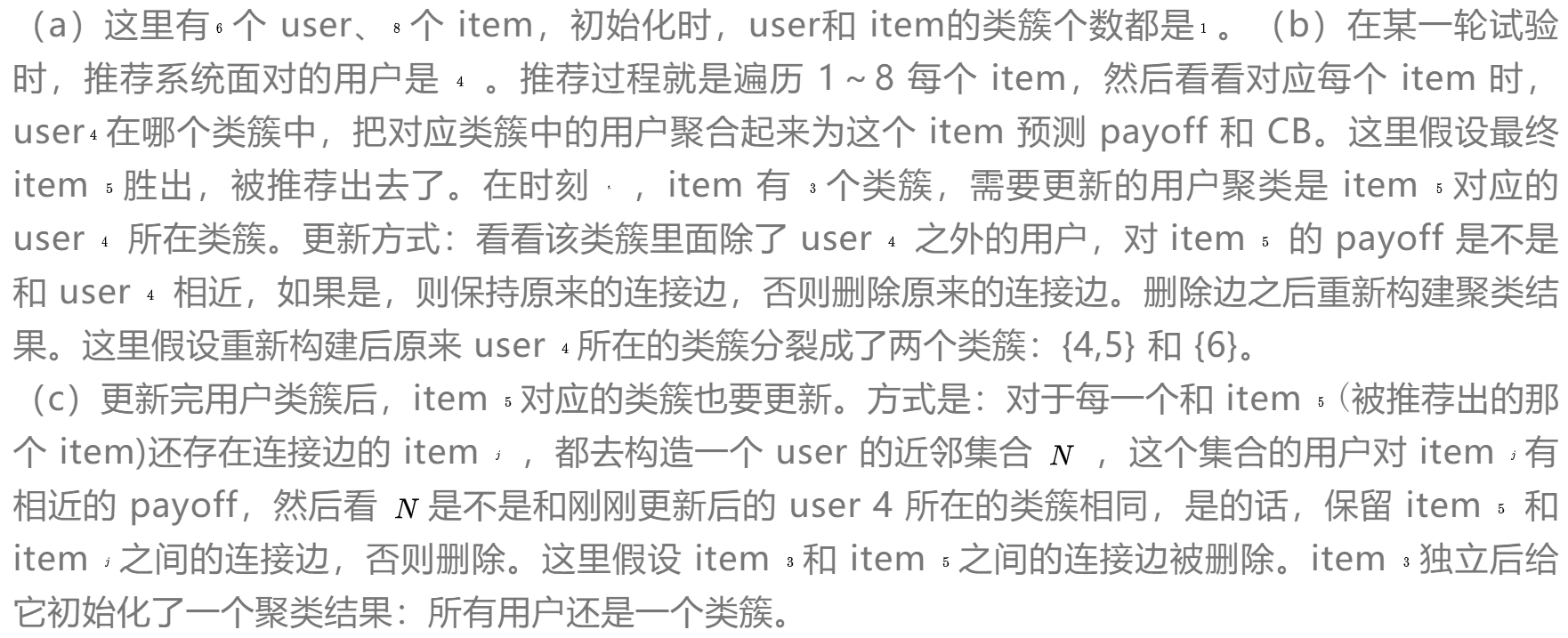
推荐系统中最经典的算法莫过于协同过滤了，它的核心假设就是："物以类聚，人以群分"，而解决EE问题的核心是：基于收益最大化寻找长短期兴趣的tradeoff，bandit算法是解决EE问题非常好的方式，而在上一节中LinUCB提出了结合UCB和上下文解决EE问题，显然其根本思想在于user&item上下文信息是对Bandit补充。在这个理念下，基于Bandit算法与协同过滤结合的方法COFIBA被提了出来。

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，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6个user、8个item，初始化时，user和item的类簇个数都是1。（b）在某一轮试验时，推荐系统面对的用户是4 。推荐过程就是遍历1～8每个item，然后看看对应每个item时，user 4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 5胜出，被推荐出去了。在时刻，item有3个类簇，需要更新的用户聚类是item 5对应的user 4所在类簇。更新方式：看看该类簇里面除了 user 4之外的用户，对 item 5的payoff是不是和user 4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 4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 5对应的类簇也要更新。方式是：对于每一个和item 5(被推荐出的那个item)还存在连接边的 item j，都去构造一个user的近邻集合N，这个集合的用户对item j有相近的payoff，然后看N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 5和 item j之间的连接边，否则删除。这里假设 item 3和 item 5之间的连接边被删除。item 3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

# 4 其他思考

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。

# 5小结

Exploration and Exploitation（EE问题，探索与开发）是计算广告和推荐系统里常见的一个问题，在给用户进行物品推荐时，不仅要投其所好，还要进行适当的长尾物品挖掘。

在权衡Exploration and Exploitation时，不妨有策略的试一试，把选择的机会给“确定好的”和“还不确定的”。

本文介绍了朴素Bandit算法、Epsilon-Greedy算法、Thompson sampling算法以及UCB算法等4个传统Bandit算法的核心思想和实现步骤。

实际上，Bandit算法是一种不太常用的推荐系统算法，究其原因，是它能同时处理的物品数量不能太多。但是，在冷启动和处理EE问题时，Bandit算法简单好用，值得一试。

# 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>

<https://mp.weixin.qq.com/s/jnpfOOrtodO0sonjVHfU6w>

<https://mp.weixin.qq.com/s/Xv3XvMkIeRPYY9KG5qafow>