推荐系统的EE问题及Bandit算法 (http://x-algo.cn/index.php/2016/12/15/ee-problem-and-bandit-algorithm-for-recommender-systems/)

2016-12-15 分类：推荐系统 (http://x-algo.cn/index.php/category/rcm/) 阅读(1879) 评论(0)

经常听身边的人说起使用推荐系统的感受，“某宝某东就是看什么给推什么”，使用者对推荐系统产生厌倦；也有听做推荐系统的同学抱怨推荐的冷启动问题，如何去尝试新用户的兴趣点，尝试到什么时候地步才算真正掌握了用户的兴趣，用户的兴趣发生改变如何灵活的调整推荐策略。这些，都与今天聊到的E&E问题有关，而Bandit算法是解决E&E问题的一种思路。本文首先描述E&E问题的及策略框架，然后介绍几个代表的Bandit算法。

E&E问题简介

问题描述

- 条件：假设我们有K个准备推荐的item，每个item的回报的服从不同的概率分布p_item，且分布参数未知
- 目标：如果有T次机会推荐，如何制定决策过程从而获取最大的累积回报

See also：多臂赌博机问题 (https://en.wikipedia.org/wiki/Multi-armed_bandit)
（Multi-armed bandit problem, K-armed bandit problem, MAB）

表现形式

- 随机式(stochastic bandit): item的回报服从某个固定的概率分布
- 对抗式(adversarial bandit): item的回报会动态调整，让你选的奖励变低

推荐应用

计算机广告和推荐系统中，有很多问题可以抽象为E&E问题：

- 冷启动问题：假设一个用户对不同类别的内容感兴趣程度不同，那么我们的推荐系统初次见到这个用户时，怎么快速地知道他对每类内容的感兴趣程度？
- 推荐item选择：假设资源池有若干item（或若干类item），怎么知道该给每个用户展示哪个（类），从而获得最大的点击？是每次都挑效果最好那个么？那么新广告如何才有出头之日？

热门标签

CRF (5) (http://x-	Tensorflow (3)	CNN (3)
中文分词 (2)	word2vec (2)	最优化 (2)
learning to rank	GPU (2)	LSTM (2)
OCR (2)	词性表 (1)	搜索 (1)
iis (1) (http://x-	HMM (1)	隐马 (1)
隐马尔科夫模型	EM (1) (http://x-	CRF词性标注 (1)
词性标注 (1)	实体识别 (1)	地名实体识别 (1)
CRF实体识别 (1)	句法分析 (1)	CRF句法分析 (1)
绘制句法树 (1)	js (1) (http://x-	css (1) (http://x-
html (1)	postaging (1)	nlp (1) (http://x-

分类目录

- Deep Learning (http://x-algo.cn/index.php/category/deep-learning/) (26)
  - CNN (http://x-algo.cn/index.php/category/deep-learning/cnn/) (11)
  - LSTM (http://x-algo.cn/index.php/category/deep-learning/lstm/) (2)
  - RNN (http://x-algo.cn/index.php/category/deep-learning/rnn/) (3)
  - 语音识别 (http://x-algo.cn/index.php/category/deep-learning/yuyinshibie/) (3)
- GPU编程 (http://x-algo.cn/index.php/category/gpucoding/) (4)
- NLP (http://x-algo.cn/index.php/category/nlp/) (12)
- 图像 (http://x-algo.cn/index.php/category/tuxiang/) (11)
- 推荐系统 (http://x-algo.cn/index.php/category/rcm/) (7)
- 数学 (http://x-algo.cn/index.php/category/%e6%95%b0%e5%ad%a6/) (1)
  - 线性代数 (http://x-algo.cn/index.php/category/%e6%95%b0%e5%ad%a6/%e7%ba%bf%e6%80%a7%e4%bb%a3%e6%95%b0/) (1)
- 数据结构 (http://x-algo.cn/index.php/category/datastruct/) (1)
  - 树 (http://x-algo.cn/index.php/category/datastruct/tree/) (1)
- 最优化 (http://x-algo.cn/index.php/category/opt/) (10)

- 推荐策略选择：假设我们有了新的推荐策略，有没有比A/B test更快的方法知道它和旧模型相比谁更靠谱？
- 其他：.....

E&E算法框架

Exploitation & Exploration

（1）Exploitation：基于已知最好策略，开发利用已知具有较高回报的item（贪婪、短期回报）

1. Advantage：充分利用已知高回报item
2. Disadvantage：陷于局部最优，错过潜在更高回报item的机会

（2）Exploration：不考虑曾经的经验，勘探潜在可能高回报的item（非贪婪、长期回报）

1. Advantage：发现更好回报的item
2. Disadvantage：充分利用已有高回报item机会减少（如已经找到最好item）

（3）目标：要找到Exploitation & Exploration的trade-off，以达到累计回报最大化。

极端情况下，Exploitation每次选择最高mean回报的item（太“confident”），Exploration每次随机选择一个item（太不“confident”），两个极端都不能达到最终目标，因此，在选择item时，我们不仅要考虑item的mean回报，同事也要兼顾confidence。

评估指标Regret

面对多个item选择，我们的选择到底有多遗憾？如何衡量解决E&E问题的好坏呢？

定义累积遗憾（regret）：

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

其中，这里假定选择每个item的回报为伯努利回报，wB(i)是第i次试验时被选中item的期望回报， w_opt是所有item中的最佳选择item的回报。累计遗憾为T次的选择的遗憾累计，每次选择的遗憾为最优选择回报与本次选择回报之差。

Bandit算法实现

naive Algorithm

最简单直接的，我们可以通过简单观察法（naive Algorithm）选取item：先对每个item进行一定次数（如100次）选择尝试，计算item的回报率，接下来选择回报率高的item。算法简单直接，但存在以下问题：

- item很多导致获取item回报率的成本太大
- 尝试一定次数（如100次）得到的“高回报”item未必靠谱
- item的回报率有可能会随时间发生变化：好的变差、差的变好

ε-Greedy

选一个(0,1)之间较小的数ε，每次决策以概率ε去勘探Exploration，1-ε的概率来

- 杂 (http://x-algo.cn/index.php/category/%e6%9d%82/) (1)
- 模型 (http://x-algo.cn/index.php/category/model/) (17)
- 算法 (http://x-algo.cn/index.php/category/algo/) (7)
  - 动态规划 (http://x-algo.cn/index.php/category/algo/dp/) (1)
  - 搜索 (http://x-algo.cn/index.php/category/algo/sousuo/) (2)
  - 语言入门 (http://x-algo.cn/index.php/category/algo/yuyanrumen/) (2)
  - 贪心算法 (http://x-algo.cn/index.php/category/algo/tanxinsuanfa/) (2)
- 计算广告 (http://x-algo.cn/index.php/category/adalgo/) (1)
- 词表 (http://x-algo.cn/index.php/category/word_dic/) (2)
- 读书 (http://x-algo.cn/index.php/category/read/) (7)
  - 强化学习 (http://x-algo.cn/index.php/category/read/qianghuaxuexi/) (5)
  - 线性代数 (http://x-algo.cn/index.php/category/read/line-algebra/) (2)
- 调度系统 (http://x-algo.cn/index.php/category/schedule/) (1)

文章归档

2017年五月 (http://x-algo.cn/index.php/2017/05/)
2017年四月 (http://x-algo.cn/index.php/2017/04/)
2017年三月 (http://x-algo.cn/index.php/2017/03/)
2017年二月 (http://x-algo.cn/index.php/2017/02/)
2017年一月 (http://x-algo.cn/index.php/2017/01/)
2016年十二月 (http://x-algo.cn/index.php/2016/12/)
2016年十一月 (http://x-algo.cn/index.php/2016/11/)
2016年九月 (http://x-algo.cn/index.php/2016/09/)
2016年八月 (http://x-algo.cn/index.php/2016/08/)
2016年七月 (http://x-algo.cn/index.php/2016/07/)
2016年六月 (http://x-algo.cn/index.php/2016/06/)
2016年四月 (http://x-algo.cn/index.php/2016/04/)
2016年三月 (http://x-algo.cn/index.php/2016/03/)
2016年二月 (http://x-algo.cn/index.php/2016/02/)

开发Exploitation，基于选择的item及回报，更新item的回报期望（见“增量更新期望回报”），不断循环下去。

这样做的好处在于：

- 能够应对变化：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
- 可以控制对Exploration和Exploitation的偏好程度：ε大，模型具有更大的灵活性（能更快的探索潜在可能高回报item，适应变化，收敛速度更快），ε小，模型具有更好的稳定性（更多的机会用来开发利用当前最好回报的item），收敛速度变慢

虽然ε-Greedy算法在Exploration和Exploitation之间做出了一定平衡，但

- 设置最好的ε比较困难，大则适应变化较快，但长期累积回报低，小则适应变好的能力不够，但能获取更好的长期回报。
- 策略运行一段时间后，我们对item的好坏了解的确定性增强，但仍然花费固定的精力去exploration，浪费本应该更多进行exploitation机会
- 策略运行一段时间后，我们已经对各item有了一定程度了解，但没用利用这些信息，仍然不做任何区分地随机exploration（会选择到明显较差的item）

Epsilon-Greedy算法的变体：

1）ε-first strategy：首先进行小部分次数进行随机尝试来确定那些item能获得较高回报，然后接下来大部分次数选择前面确定较高回报的item

2）ε-decreasing strategy: epsilon随着时间的次数逐步降低，开始的时候exploration更多次数，然后逐步降低（ε_t=1/log(itemSelectCnt+0.00001)），增加exploitation比重(Annealing-Epsilon-Greedy)

增量更新item的回报期望：

由于每次决策都是基于各item的历史回报均值，那么随着决策次数的增多，存储历史回报的空间消耗会越来越大，计算历史回报平均值的计算量也会越来越大，为了提高效率，对于历史回报平均值的计算可以做一下变形：

$$
\begin{aligned}
Q_{k+1} &= \frac{1}{k+1}\sum_{i=1}^{k+1} r_i \\
&= \frac{1}{k+1}\left(r_{k+1} + \sum_{i=1}^{n} r_i\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + kQ_k + Q_k - Q_k\right) \\
&= \frac{1}{k+1}\left(r_{k+1} + (k+1)Q_k - Q_k\right) \\
&= Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right],
\end{aligned}
$$

SoftMax

ε-Greedy在探索时采用完全随机的策略，经常会选择一个看起来很差的item，此问题的解决方法之一是，基于我们目前已经知道部分item回报信息，不进行随机决策，而是使用softmax决策找出回报最大的item。

- Idea:

$$\mathbb{P}(\text{arm 1}) = \frac{\hat{\mu_1}}{\hat{\mu_1} + \hat{\mu_2}}$$
$$\mathbb{P}(\text{arm 2}) = \frac{\hat{\mu_2}}{\hat{\mu_1} + \hat{\mu_2}}$$

- Variant:

$$\mathbb{P}(\text{arm 1}) = \frac{e^{\frac{\hat{\mu_1}}{\tau}}}{}$$

$$\mathbb{P}(\text{arm 2}) = \frac{e^{\frac{\mu_2}{T}}}{e^{\frac{\mu_1}{T}} + e^{\frac{\mu_2}{T}}}$$

$$T \to \infty : \text{Pure exploration}$$
$$T = 0 : \text{Pure exploitation}$$

其中，T为温度参数，温度高则选择各item的概率趋于随机，温度低则以更确定性的概率选择当前平均回报最好的item（greedy）。

SoftMax利用softmax函数来确定各item的回报的期望概率排序，进而在选择item时考虑该信息，减少exploration过程中低回报率item的选择机会，同时收敛速度也会较ε-Greedy更快。

但是，softmax算法的缺点是：

- 没有考虑预估item回报率期望的置信度信息，因而选择的高回报率item未必靠谱。
- 无法事先知道t的大小，因问题而异，同时也不容易与其他算法直接比较

Softmax变体：Annealing-Softmax:

随着策略的运行，我们期望降低T温度参数以减小\exploration所占的比
例：T=1/log(itemSelectCnt+0.0000001)

UCB, Upper Confidence Bound

统计学中，我们使用置信区间来度量估计的不确定性/置信性。如我们摇骰子一次得到的点数为2，那么得到均值的估计也是2（实际平均点数是3.5），但显然这个估计不太靠谱，可以用置信区间量化估计的变化性：骰子点数均值为2，其95%置信区间的上限、下限分别为1.4、5.2。

UCB思想是乐观地面对不确定性，以item回报的置信上限作为回报预估值的一类算法，其基本思想是：我们对某个item尝试的次数越多，对该item回报估计的置信区间越窄、估计的不确定性降低，那些均值更大的item倾向于被多次选择，这是算法保守的部分（exploitation）；对某个item的尝试次数越少，置信区间越宽，不确定性较高，置信区间较宽的item倾向于被多次选择，这是算法激进的部分（exploration）。

其计算item期望的公式：

$$\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

其中，x_j是item_j的平均回报，n_j是item_j截至当前被选择的次数，n为当前选择所有item的次数。上式反映了，均值越大，标准差越小，被选中的概率会越来越大，起到了exploit的作用；同时哪些被选次数较少的item也会得到试验机会，起到了explore的作用。

与ε-Greedy算法、softmax算法相比，这种策略的好处在于：

- 考虑了回报均值的不确定性，让新的item更快得到尝试机会，将探索+开发融为一体
- 基础的UCB算法不需要任何参数，因此不需要考虑如何验证参数（ε如何确定）的问题

UCB1算法的缺点：

- UCB1算法需要首先尝试一遍所有item，因此当item数量很多时是一个问题
- 一开始各item选择次数都比较少，导致得到的回报波动较大（经常选中实际比较差的item）

UCB1算法描述

Deterministic policy: UCB1.
Initialization: Play each machine once.
Loop:

- Play machine j that maximizes $\bar{x}_j + \sqrt{\frac{2\ln n}{n_j}}$, where $\bar{x}_j$ is the average reward obtained from machine j, n_j is the number of times machine j has been played so far, and n is the overall number of plays done so far.

UCB2算法描述

基于UCB1的思想进一步改进，UCB2使得累计遗憾regret界线更小

Deterministic policy: UCB2.
Parameters: $0 < \alpha < 1$.
Initialization: Set $r_j = 0$ for $j = 1, \ldots, K$. Play each machine once.
Loop:

1. Select machine j maximizing $\bar{x}_j + a_{n,r_j}$, where $\bar{x}_j$ is the average reward obtained from machine j, a_{n,r_j} is defined in (3), and n is the overall number of plays done so far.
2. Play machine j exactly $\tau(r_j + 1) - \tau(r_j)$ times.
3. Set $r_j \leftarrow r_j + 1$.

其中，

$$a_{n,r} = \sqrt{\frac{(1+\alpha)\ln(en/\tau(r))}{2\tau(r)}}$$

where

$$\tau(r) = \lceil (1+\alpha)^r \rceil.$$

LinUCB

上述算法均没用充分利用上下文信息Contextual，而LinUCB的基本思想是对每个item的回报估计及其置信区间同时建模，然后每次选择回报的估计值与其标准差的和最大的那个item，因此LinUCB在推荐系统中，能够较好地平衡显示用户已经喜欢的某类文章和对其他没怎么看过的类别的文章，从而引导用户对未知类别的探索。

- User特征：人口统计学特征、地理未知特征、行为偏好特征.....
- item特征：URL类别、主题分类、流行度、新颖度.....

Algorithm 1 LinUCB with disjoint linear models.

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

trarily, 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

基于上下文的LinUCB，与普通的UCB相比

- 由于加入了特征，其收敛速度比UCB更快，特征工程越好，提升效果越好
- 由于参与计算的是特征，所以可以处理动态的推荐资源池
- 特征较多时，影响计算效率，需要降维处理

Thompson sampling

UCB算法部分使用概率分布（仅置信区间上界）来量化不确定性。而Thompson sampling基于贝叶斯思想，全部用概率分布来表达不确定性。

假设每个item有一个产生回报的概率p，我们通过不断试验来估计一个置信度较高的概率p的概率分布。如何估计概率p的概率分布呢？ 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item都维护一个beta分布的参数。每次试验选中一个item，有回报则该item的wins增加1，否则lose增加1。每次选择item的方式是：用每个item现有的beta分布产生一个随机数b，选择所有item产生的随机数中最大的那个item。

Algorithm 2 Thompson sampling for the Bernoulli bandit
Require: α, β prior parameters of a Beta distribution
$S_i = 0, F_i = 0, \ \forall i$. {Success and failure counters}
for $t = 1, \ldots, T$ **do**
  for $i = 1, \ldots, K$ **do**
    Draw θ_i according to Beta$(S_i + \alpha, F_i + \beta)$.
  end for
  Draw arm $\hat{\imath} = \arg\max_i \theta_i$ and observe reward r
  if $r = 1$ **then**
    $S_{\hat{\imath}} = S_{\hat{\imath}} + 1$
  else
    $F_{\hat{\imath}} = F_{\hat{\imath}} + 1$
  end if
end for

相比于UCB算法，Thompson sampling：

- UCB采用确定的选择策略，可能导致每次返回结果相同（不是推荐想要的），Thompson Sampling则是随机化策略。
- Thompson sampling实现相对更简单，UCB计算量更大（可能需要离线/异步计算）
- 在计算机广告、文章推荐领域，效果与UCB不相上下或更好competitive to or better
- 对于数据延迟反馈、批量数据反馈更加稳健robust

参考文献

- https://en.wikipedia.org/wiki/Multi-armed_bandit (https://en.wikipedia.org/wiki/Multi-armed_bandit)
- http://engineering.richrelevance.com/bandits-recommendation-systems/ (http://engineering.richrelevance.com/bandits-recommendation-systems/)
- An Empirical Evaluation of Thompson Sampling
- Finite-time Analysis of the Multiarmed Bandit Problem
- Bandit Algorithms for Website Optimization
- A Survey on Contextual Multi-armed Bandits
- Thompson Sampling for Contextual Bandits with Linear Payoffs
- A Contextual-Bandit Approach to Personalized News Article

Recommendation

- 推荐系统的苟且和远方
- 专治选择困难症——bandit算法
- UCB算法升职记——LinUCB算法

分享到：

相关推荐

- ListNet原理 (http://x-algo.cn/index.php/2016/08/18/listnet-principle/)
- LambdaRank和LambdaMART原理 (http://x-algo.cn/index.php/2016/08/16/1026/)
- RankSVM原理 (http://x-algo.cn/index.php/2016/08/09/ranksvm/)
- RankNet算法原理和实现 (http://x-algo.cn/index.php/2016/07/31/ranknet-algorithm-principle-and-realization/)
- Wand算法 (http://x-algo.cn/index.php/2016/07/13/812/)
- word2vec在工业界的应用场景 (http://x-algo.cn/index.php/2016/03/12/281/)

评论 抢沙发

你的评论可以一针见血

验证码*

提交评论

昵称

昵称 (必填)

邮箱

邮箱 (必填)

网址

网址