



Multi-Armed Bandit: UCB (Upper Bound Confidence)



冯伟
Hulu推荐算法

已关注

王喆等 49 人赞同了该文章

上一讲主要内容回顾

假设我们开了一家叫 Surprise Me 的饭馆

- 客人来了不用点餐，由算法从 N 道菜中选择一道菜推荐给客人
- 每道菜都有一定的失败概率：以 $1-p$ 的概率不好吃，以 p 的概率做得好吃
- 算法的目标是让满意的客人越多越好。

解决方法：

ϵ -greedy 算法：

- 以 ϵ 的概率从 N 道菜中随机选择(概率为 $\frac{\epsilon}{N}$)一个让客人试吃
- 以 $1 - \epsilon$ 的概率选择 N 道菜中选择好吃的概率最高的菜推荐给客

充分利用历史信息进行选择

ϵ -greedy 生硬的将选择过程分成探索阶段 (Exploration) 和 利用阶段 (Exploitation)，在探索时对 所有物品进行以同样的概率(概率为 $\frac{\epsilon}{N}$) 进行探索，并不会利用任何历史信息，包括 (1) 某道菜被探索的次数， (2) 某道菜获得好吃反馈的比例。

让我们忘记探索阶段和利用阶段，仔细想想如何利用历史信息，找到最值得被推荐的菜：

观测 1: 如果一道菜已经推荐了 k 遍 (获取了 k 次反馈)，我们就可以算出菜做的好吃的概率：

$$\hat{p} = \frac{\sum \text{reward}_i}{k}$$

当 k 趋近正无穷时， \tilde{p} 会趋近于真实的菜做的好吃的概率 p

观测 2: 现实当中一道菜被试吃的次数 k 不可能无穷大，因此估计出的好吃的概率 \tilde{p} 和真实的好吃的概率 p 总会存在一个差值 Δ ，即 $\tilde{p} - \Delta \leq p \leq \tilde{p} + \Delta$

基于上面两个观测，我们可以定义一个新的策略：每次推荐时，总是乐观地认为每道菜能够获得的回报是 $\tilde{p} + \Delta$ ，这便是著名的Upper Confidence Bound (UCB) 算法，代码如下所示。

```
def UCB(t, N):
    upper_bound_probs = [avg_rewards[item] + calculate_delta(t, item) for item in range(N)]
    item = np.argmax(upper_bound_probs)
    reward = np.random.binomial(n=1, p=true_rewards[item])
    return item, reward

for t in range(1, T): # T个客人依次进入餐馆
    # 从N道菜中推荐一个, reward = 1 表示客人接受, reward = 0 表示客人拒绝并离开
    item, reward = UCB(t, N)
    total_reward += reward # 一共有多少客人接受了推荐
```

真实的概率和估计的概率之间的差值 Δ

最后只需解决一个问题，真实的概率和估计的概率之间的差值 Δ 到底怎么计算呢？

在进入公式之前，让我们直观的理解影响 Δ 的因素：

- 对于被选中的菜，多获得一次反馈会使 Δ 变小，最终会小于其他没有被选中的菜
- 对于没被选中的菜， Δ 会随着轮数的增大而增大，最终会大于其他被选中的菜

下面我们正式介绍如何计算 Δ ，首先介绍Chernoff-Hoeffding Bound：

[Chernoff-Hoeffding Bound] 假设 $reward_1, \dots, reward_n$ 是在 $[0, 1]$ 之间取值的独立同分布随机变量，用 $\tilde{p} = \frac{\sum_i reward_i}{n}$ 表示样本均值，用 p 表示分布的均值，那么有

$$P\{|\tilde{p} - p| \leq \delta\} \geq 1 - 2e^{-2n\delta^2}$$

当 δ 取值为 $\sqrt{2\ln T/n}$ 时 (其中 T 表示有 T 个客人， n 表示菜被吃过的次数)，可以得到

$$P\{|\tilde{p} - p| \leq \sqrt{2\ln T/n}\} \geq 1 - \frac{2}{T^4}$$

也就是说 $\tilde{p} - \sqrt{2\ln T/n} \leq p \leq \tilde{p} + \sqrt{2\ln T/n}$ 是以 $1 - \frac{2}{T^4}$ 的概率成立的：

- 当 $T=2$ 时，成立的概率为0.875
- 当 $T=3$ 时，成立的概率为0.975
- 当 $T=4$ 时，成立的概率为0.992

可以看出 $\Delta = \sqrt{2\ln T/n}$ 是一个不错的选择。

最后，我们附上完整的代码，跟第一讲不一样的地方已经重点加粗标注。

```
import numpy as np

T = 1000 # T个客人
```

```
N = 10 # N道菜

true_rewards = np.random.uniform(low=0, high=1, size=N) # 每道菜好吃的概率
estimated_rewards = np.zeros(N) # 每道菜好吃的估计概率
chosen_count = np.zeros(N) # 各个菜被选中的次数
total_reward = 0

def calculate_delta(T, item):
    if chosen_count[item] == 0:
        return 1
    else:
        return np.sqrt(2 * np.log(T) / chosen_count[item])

def UCB(t, N):
    upper_bound_probs = [estimated_rewards[item] + calculate_delta(t, item) for i
    item = np.argmax(upper_bound_probs)
    reward = np.random.binomial(n=1, p=true_rewards[item])
    return item, reward

for t in range(1, T): # T个客人依次进入餐馆
    # 从N道菜中推荐一个, reward = 1 表示客人接受, reward = 0 表示客人拒绝并离开
    item, reward = UCB(t, N)
    total_reward += reward # 一共有多少客人接受了推荐

    # 更新菜的平均成功概率
    estimated_rewards[item] = ((t - 1) * estimated_rewards[item] + reward) / t
    chosen_count[item] += 1
```

欢迎订阅微信公众号 "零基础机器学习", 搜索微信号: ml-explained

编辑于 2018-01-01

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

[机器学习](#) [在线机器学习](#) [强化学习 \(Reinforcement Learning\)](#)

文章被以下专栏收录



零基础机器学习

已关注

推荐阅读



手把手教你从零起步构建自己的图像搜索模型

AI研习社

发表于AI研习社...

在线学习 (Online Learning) 第一讲

简介 在线预测可以被看做一个在玩家（算法）和环境之间的重复游戏。假设 T 为游戏的总轮数。那么在每一轮 $t(t=1, \dots, T)$ 上，在玩家和环境之间的这个游戏可以直观地表示为： 环境：从问题...

看看

高级强化学习系列 第一讲 联合模型的强化学习算法（三）

时间过得飞快，转眼间一周又过去了。今天继续分享联合模型的强化学习算法。前面我们介绍了指导策略搜索方法（GPS）。GPS利用模型来指导策略搜索，其优点是用来做指导的模块是轨迹最优方法，...

天津包子馅儿

增
本
目
作
列
地
址

14 条评论

[切换为时间排序](#)

写下你的评论...



王喆

1 年前

啥也别说了，抓紧把这几篇文章收录到咱Albook里面：)

赞



冯伟 (作者) 回复 王喆

1 年前

哈哈，被你发现了

赞



Jiaxi Tang

1 年前

Chernoff-Hoeffding Bound 那里有错误 应该是大于等于 $1 - 2e^{(-2nt^2)}$

赞



冯伟 (作者) 回复 Jiaxi Tang

1 年前

非常感谢，我已经在原文中进行了修正，欢迎继续指正：)

赞



Kyle Huang

11 个月前

最后更新菜的平均成功概率, 为什么用 t , 而不是像上一讲的code用 `chosen_count[item]`

2



rockguy 回复 Kyle Huang

7 个月前

感觉应该用 `chosen_count[item]`

赞



生如夏花 回复 rockguy

6 个月前

同意你们俩，仍然用 `average-sampling method` 估计奖赏，用 `chosen_count[item]`

赞

[展开其他 2 条回复](#)

鸣啦啦啦

4 个月前

其实思路就是：认为 $(p_head + \delta)$ 就是真实的 p ，那么自然我们采用 $\max(p_head + \delta)$ 的 `item` 作为每次推荐的菜。

“充分利用历史信息进行选择”和“Upper Bound Confidence名称的由来”都体现在

$\max(p_{\text{head}} + \delta)$ 公式里了。

👍 2



哈哈哒哒 回复 鸣啦啦啦

4 个月前

层主评论的很简洁易懂，赞一个！

还有一个问题不明白：为什么根据 $\max(p_{\text{head}} + \delta)$ 选出概率最大的菜之后还要做一次采样？

```
np.random.binomial(n=1, p=true_rewards[item])
```

👍 赞



Zepp 回复 哈哈哒哒

3 个月前

我觉得是要生成新客人的reward

👍 赞



Mimo 回复 哈哈哒哒

2 个月前

每个客人对于所选菜的随机反应吧，其实也就是reward

👍 赞



henryWang 回复 哈哈哒哒

1 个月前

因为概率最大的菜的概率不是1呀，再好的菜也有人不爱吃啊

👍 赞

▲ 赞同 49 ▼

💬 14 条评论

➦ 分享

★ 收藏

...

