工作中高优算法总结

zhangliang605*

DATOUBANG.INC

June 7, 2024

*电子邮件: zhangliang605@gmail.com

这个文档是我工作中遇到的各种算法的总结，本意是希望能对自己的知识体系进行系统性的总结归纳。随着工作时间越来越长，渐渐地，我有一种比较强烈的感觉，所谓”吾生也有涯，而知无涯”。一味地追求知识固然是一件正确的事情，但学习的同时也要特别关注创造。一个知识点会被很多人学习，但是创造往往是少数人能做到的事情。

因此，这个文档也寄托了我希望能有些创造性的产品或者想法产生的期望。

关键字：整数规划，背包问题，营销出价，广告定价，流量控制，数据结构算法

目录

1 PID 算法在成本控制领域的应用

常见的成本控制方法可分为认为干预和算法自动控制两种。顾名思义，人为干预是通过人工实时监控广告投放情况，当发现实际成本低于或超出预期预算时，通过人工调整广告出价或修改人群定向等方式调节投放花费；算法自动控制是指采用相关算法，监控投放成本，并根据异常自动调节广告出价，达到控制成本的目的。

1.1 PID 控制算法简介

PID 算法包含了比例（Proportion）、积分（Integration）、微分（Differentiation）三个环节，其根据被控对象实际输出与目标值的偏差，按照三个环节进行运算，最终达到稳定系统的目的。

简答的说, k_p 代表现在, k_i 代表过去, k_d 代表未来。在实际应用中还是需要考虑具体参数大小, 可以通过 grid search, 根据相应时间、超调量、稳态误差指标, 来综合考虑 PID 值。

PID 调价也存在着一些缺陷，简单泛化能力强式优点也是缺点，只需要根据设定 cpc 和实际 cpc 的反馈就能够调节。但是，在某些固定场景下，cpc 的波动会呈现固定的 pattern，例如在某几个小时流量指令非常好，cpc 会特别低，这就需要使用机器学习来记忆到哪些 campaingn 在哪些时间点需要提高价格，使用强化学习出价在充分利用投放数据、建立 MDP 模型、序列号决策这些方面就有了天然优势。

PID 具体公式如下：

$$
\begin{aligned}
err_t &= target_{cpc_t} - real_{cpc_t} \\
\Delta_t &= k_p(err(t) + \frac{1}{k_i}\int err(t)\mathrm{d}t + k_d\frac{\mathrm{d}err(x)}{\mathrm{d}t}) \\
\lambda_{t+1} &= \lambda_t + \Delta_t
\end{aligned}
\tag{1}
$$

其中：

- err_t: 第 t 轮 PID 的误差值
- $real\_cpc_t$: 第 t 轮 PID 的实际值
- $target\_cpc_t$: 第 t 轮 PID 的目标值
- k_p: 比例增益

- k_i: 积分时间常数
- k_d: 微分时间常数
- Δ: 第 t 轮 PID 的增量系数
- λ_t: 第 t 轮 PID 的调控系数
- λ_{t-1}: 第 t–1 轮 PID 的调控系数

关于 P k_p 是比例系数，假设目标 cpc 0.4，实际 cpc 0.2，误差是 0.2，k_p 越大，反应幅度就会越大，新的 λ 就会增加很多，出价就会增加很多，但是 k_p 不能够过大，不然会导致超额调整，出价过高。所以，k_p 代表了根据当前误差反应的比例。

关于 I k_p 的存在是为了解决稳态误差。
假如当前 cpc 偏低，每个小时都提高价格，但是市场价格（出价第二高的广告主出价）也在下降，所以，虽然每个小时都按照 PID 调控系数提价，但是由于市场价格在降价，导致基于 PID 的每次提价都没有提上去。像这种如果一直存在，我们称之为稳态误差。积分的存在就是通过过去差值的经验来调整出价，来消除这个稳态误差。

但是，实际投放过程中基本不会存在这样的稳态，因为竞价系统是动态的，只能说市场价格可能随着时间有些固定的变化，但是变化不一定式稳定方向，所以 k_i 值在实际使用中需要慎重，如果设置特别大，会导致上个小时已经不存在的误差，影响到当前小时，所以 k_i 即使要使用，最好设置的非常小。

关于 D k_d 项经常被称为微分项，当两次调控间隔十分小，$(err_t - err_{t-1})/1$ 计算的就是斜率，如果间隔十分小，那么这个斜率就可以一定程度体现次 err 的走向，这也是为社么说微分项代表未来。但是，如果两次间隔十分大、或者噪音非常多，微分项的作用就不大了。对于 1 小时调控一次的 PID 调价，k_d 项可以为 0。实际上，很多 PID 控制器仅用 k_p 和 k_i 就已经足够了。

1.2 PID 算法简单代码

PID 控制算法可以分为位置时 PID 和增量式 PID 控制算法。两者的区别：

- (1) 位置式 PID 控制的输出与整个过去的状态有关，用到了误差的累加值。而增量式 PID 的输出只与当前拍和前两拍的误差有关，因此位置式 PID 控制的累计误差相对更大。

- (2) 增量式 PID 控制输出的是控制量增量，如果计算机出现故障，误动作影响较小，而执行机构本身有记忆功能，仍可保持原位，不会严重影响系统的工作，而位置式的输出直接对应对象的输出，因此对系统影响较大。

1.2.1 位置式 PID

$$u(k) = K_P e(k) + K_I \sum_{i=0} e(i) + K_D[e(k) - e(k-1)] \tag{2}$$

```python
# 位置型PID
def pid_positional(budget_target, budget_error):
    P = budget_error[0]
    I = sum(Budget_error)
    D = budget_error[0] - Budget_error[1]

    return kp * P + ki * I + kd * D
```

1.2.2 增量式 PID

$$\begin{aligned}\Delta u(k) &= u(k) - u(k-1) \\ &= K_p[e(k) - e(k-1)] + K_I e(k) + K_D[e(k) - 2e(k-1) + e(k-2)]\end{aligned} \tag{3}$$

```python
import pandas as pd
import random
import matplotlib.pyplot as plt

# 模拟调控N天，第1天无法用PID，从第2天开始施加PID调控
N = 10

# 每天目标预算
Budget_target = 100

# 每天实际花出预算
```

```python
Budget_real = [0] * N                    # 历次PID调控给出的真实预算
Budget_real[1] = 80

# 调控后每天预期预算
Budget_expected_positional = [0] * N
Budget_expected_incremental = [0] * N        # 历次PID调控后预期预算
Budget_expected_incremental_u = [0] * N      # 历次PID调控后预算差值 = 预期预算
                                        - 目标预算

# PID系数
kp = 0.5
ki = 0.5
kd = 0.5

# 积分天数
T = 3

# 增量型PID
def pid_incremental(Budget_target, Budget_error, u):

  delta = kp * (Budget_error[0] - Budget_error[1]) + ki * Budget_error[0] + kd
                                        * (Budget_error[0] - 2 *
                                        Budget_error[1] + Budget_error[2])

  return delta + u

# 模拟每天实际花出预算，在目标预算和预期预算之间
def get_budget(Budget_min, Budget_max):
  return random.uniform(Budget_min, Budget_max)

# 第2天开始PID调控
for i in range(2, N):
  Budget_error = []
  for j in range(i-1, max(0, i-T), -1):
    Budget_error.append(Budget_target - Budget_real[j])
  if len(Budget_error) < T:
    Budget_error = Budget_error + [0] * (T - len(Budget_error))

  # 增量型PID
```

```
Budget_expected_incremental_u[i] = pid_incremental(Budget_target,
                                       Budget_error,
                                       Budget_expected_incremental_u[i-1])
Budget_expected_incremental[i] = Budget_target +
                                       Budget_expected_incremental_u[i]
Budget_min = min(Budget_target, Budget_expected_incremental[i])
Budget_max = max(Budget_target, Budget_expected_incremental[i])
Budget_real[i] = get_budget(Budget_min, Budget_max)
```

1.3 参考资料

- PID 算法在广告成本控制领域的应用
- 广告出价—如何使用 PID 控制广告投放成本

2　Uplift 算法在成本定价领域的应用

3 因果推断建模

3.1 参考资料

- 分布式因果推断在美团履约平台的探索和实践
- 因果推断之 Uplift Model|CausalML 实战篇
- CausalML: A Python Package for Uplift Modeling and Causal Inference with ML
- About CausalML

4 多约束整数规划

4.1 贪婪算法

4.2 二分算法

4.3 单纯形法

4.4 分枝界定法

4.5 启发式算法

4.6 拉格朗日乘子法

4.7 同步坐标下降法

5 ROI 公式推导

5.1 ROI 公式

$$ROI = \frac{CVR^{30} - CVR^{5}}{CVR^{30} * 30 - CVR^{5} * 5} \propto \frac{CVR^{30} - CVR^{5}}{CVR^{5}}$$

5.2 CVR 评分选择

假定选择两个价格档位，最低档 5 元，最高档 30 元（选择最高档和最低档，价格敏感性较为明显，容易学出来）。

5.3 边际 ROI 公式推导

边际 ROI 公式推导

$$\begin{aligned}
\text{边际}ROI &= \frac{bk^{30} - bk^{5}}{cost^{30} - cost^{5}} \\
&= \frac{bk^{30} - bk^{5}}{bk^{30} * 30 - bk^{5} * 5} \\
&= \frac{CVR^{30} - CVR^{5}}{CVR^{30} * 30 - CVR^{5} * 5}
\end{aligned} \tag{4}$$

分子分母同时除以曝光量 UV，假设 5 元档和 30 元档的曝光量 UV 是拉齐的，（如果不拉齐，就需要归一操作）

5.4 正比公式推导

正比公式推导

$$\begin{aligned}
\text{边际}CAC &= \frac{1}{\text{边际}ROI} \\
&= \frac{CVR^{30} * 30 - CVR^{5} * 5}{CVR^{30} - CVR^{5}} \\
&= \frac{CVR^{30} * 30 - CVR^{5} * 30 + CVR^{5} * 30 - CVR^{5} * 5}{CVR^{30} - CVR^{5}} \\
&= 30 + \frac{CVR^{5} * 25}{CVR^{30} - CVR^{5}} \\
&= 30 + \frac{CVR^{5}}{CVR^{30} - CVR^{5}} * 25
\end{aligned} \tag{5}$$

因此，$\text{边际}CAC \propto \frac{CVR^{5}}{CVR^{30} - CVR^{5}}$ 因此，$\text{边际}ROI \propto \frac{CVR^{5}}{CVR^{30} - CVR^{5}}$

$$\max \sum_{i,j} x_{i,j} \cdot bind\_card_{i,j}$$

$$s.t. \quad \sum_{i,j=1} x_{i,j} \cdot (all\_ctr_{i,j} - bytepay\_ctr_{i,j}) <= \beta \cdot |PV| \cdot all\_ctr_{emp}$$

$$\sum_{j} x_{i,j} = 1, x_{i,j} \in 0, 1$$

$$\max \sum_{i,j} x_{i,j} \cdot bind\_card_{i,j}$$

$$s.t. \qquad \sum_{i,j=1} x_{i,j} \cdot (all\_ctr_{i,j} - bytepay\_ctr_{i,j}) <= \beta \cdot |PV| \cdot all\_ctr_{emp}$$

$$\sum_{j} x_{i,j} = 1, x_{i,j} \in 0, 1$$

参考文献

[1] Wray J, Green G G R. Neural networks, approximation theory, and finite precision computation[J]. Neural networks, 1995, 8(1): 31–37.

[2] Ham F M, Kostanic I. Principles of neurocomputing for science and engineering[M]. McGraw–Hill Higher Education, 2000.