直播室停留时间的估计

曾致远，李旭浚，王之栋
清华大学计算机科学与技术系
{zengzy20,lixujun20,wang-zd19}@mails.tsinghua.edu.cn

Abstract

本文针对本次“华罗庚杯”数学建模竞赛的 A 题“直播室停留时间的估计”建立了一个数学模型，其核心是退出和进入直播间时刻的概率分布，这两个概率分布的具体计算需要用到未知的参数。对于未知参数的求解，我们使用已知的用户相邻进入直播间时刻信息进行最大似然估计；对于最终答案的求解，我们使用贝叶斯公式进行估计。经过实验验证，我们的算法不仅求出了合理的结果，同时有着很高的运行效率。

1 引言

本文的题目来源于 2020 年第二十八届清华大学“华罗庚杯”数学建模竞赛的 A 题“直播室停留时间的估计”。这个问题给定了一场体育赛事直播室的若干信息，包括每个用户每次进入直播室的时刻、每个用户每次发送弹幕和礼物的信息（时刻和金额等）、每个用户的性别和是否为 VIP 等，要求通过这些已有的数据估算每个用户每次进入直播室后的停留时长。

针对这个问题，我们建立了一种数学模型；基于这个模型，我们提出了一种简单而高效的算法。具体而言，我们首先估计了每个用户在退出直播间后重新进入直播间时刻的概率分布，然后估计了每个用户在进入直播间后退出直播间时刻的概率分布；对于这两个概率分布，一些具体的计算参数是未知的，需要通过输入数据进行计算。基于我们建模的两个分布，可以计算出每个用户在相邻两个时刻分别加入了直播间的概率，这样可以通过对该概率考虑最大似然估计以求得前面未知的参数。在参数已知进而具体化概率分布后，可以用贝叶斯公式估计处每个用户每次进入直播室后的退出时间，这和估计停留时长是等价的。

2 模型

2.1 记号约定

把所有的 n 个用户编号为 $1, 2, \cdots, n$，同时把 20:30:00 到 21:20:00 之间的每一秒钟视为一个时刻，时刻的编号为 $0, 1, \cdots, T-1$。对于第 i 个用户，记其进入直播间的次数为 E_i，每次进入直播间的时刻分别为 $\mathbf{e_{i}}_1, \mathbf{e_{i}}_2, \cdots, \mathbf{e_{i}}_{E_i}$；再记其发送弹幕和赠送付费礼物（之后

这两个被并称为一次 action）的次数为 C_i，每次 action 的发生时刻为 $\mathbf{t_{i}}1, \mathbf{t_{i}}2, \cdots, \mathbf{t_{i}}C_i$，每次 action 的金额为 $\mathbf{c_{i}}1, \mathbf{c_{i}}2, \cdots, \mathbf{c_{i}}C_i$（这里记一次弹幕对应的金额为 0）。

以上的信息均可以直接通过已有的数据得到。一个特殊的情形是，有的用户可能会在直播开始前已经进入直播间，那么会出现在（数据记录的）进入直播间时刻之前就发生 action 的情形。对于这样的用户，我们视为其在 20:30:00 进入了一次直播间。还有的用户在 21:20:00 之后进入直播间，我们无视这样的进入，认为其停留一秒钟后立刻离开。

最终的求解目标是，估计每位用户每次进入直播室后在直播室的停留时长，这等价于估计离开的时刻。这里记第 i 位用户在第 j 次（这里满足 $1 \le j \le E_i$）进入直播室后的实际离开时刻为 $\mathbf{s_{i}}j$，我们对其的估计值为 $\widetilde{\mathbf{s_{i}}j}$。

2.2 【重新进入直播间时刻】的概率分布

假设现在已知第 i 个用户在 t_1 时刻退出了直播间，且已知其在 $[t_1, T)$ 中某一时刻重新进入了直播间。我们视其实际重新进入直播间的时刻服从一个近似的指数分布。具体而言，第 $t \in [t_1, T)$ 时刻（第一次）重新进入直播间的概率 $Q_i(t_1, t)$ 正比于 $\lambda_i e^{-\lambda_i(t-t_1)}$，做一个归一化得到其完整的概率表达式 $Q_i(t_1, t) = \frac{e^{-\lambda_i(t-t_1)}}{\sum_{t' \in [t_1, T)} e^{-\lambda_i(t'-t_1)}}$，此处 λ_i 是一与用户相关的参数，我们限制其满足 $\lambda_i > 0$。

2.3 【退出直播间时刻】的概率分布

假设现在已知第 i 个用户在 t_1 时刻进入了直播间，那么其必然是在 $[t_1, T)$ 中的某一时刻退出了直播间。

我们首先对 $[0, T)$ 中的每一时刻 t 定义一个用户 i 的“热情度” $\mathbf{p_{i}}t$。从现实意义角度看，如果用户在某一时刻有一个 action，那么其应当在这一时刻热情达到了一个高潮，之后热情会随时间的推移减少；我们把热情量化为热情度，认为其服从一个近似的正态分布，这里记 $f_i(t, \mu, c) = \frac{1}{\sigma_i(c)} e^{-\frac{(t-\mu)^2}{2\sigma_i(c)^2}}$，其中 c 和 $\sigma_i(c)$ 分别是一个 action 的金额和该金额所带来正态分布的标准差，μ 是该 action 发生的时刻，t 是考虑该 action 对哪个时刻热情度的贡献。考虑热情度的叠加，最终有 $\mathbf{p_{i}}t = \sum_{\mathbf{t_{i}}j \le t} \alpha_i(\mathbf{c_{i}}j) f_i(t, \mathbf{t_{i}}j, \mathbf{c_{i}}j)$，其中 $\alpha_i(c)$ 是考虑不同 action 的金额带来不同热情度的叠加比例。

我们同时还要考虑每个时刻整个直播间的“总热情度”，我们视其为所有用户热情度的和，t 时刻的总热情度 $\mathbf{P}_t = \sum_{1 \le i \le n} \mathbf{p_{i}}t$。

最终可以用热情度和总热情度来估计用户是在某一时刻退出直播间的概率。具体而言，用户 i 在 $t \in [t_1, T)$ 时刻退出直播间的概率 $R_i(t_1, t)$ 正比于 $\frac{1}{1+\mathbf{p_{i}}t} + \beta_i \frac{1}{1+\mathbf{P}_t}$，即 $R_i(t_1, t) = \frac{\frac{1}{1+\mathbf{p_{i}}t} + \beta_i \frac{1}{1+\mathbf{P}_t}}{\sum_{t' \in [t_1, T)} (\frac{1}{1+\mathbf{p_{i}}t'} + \beta_i \frac{1}{1+\mathbf{P}_{t'}})}$，其中 β_i 是一与用户相关的参数，我们限制其满足 $\beta_i \ge 0$。

2.4 核心结论

假设现在已知第 i 个用户在 t_2 时刻（第一次）重新进入了直播间，在这之前其最后一次进入直播间是在 t_1 时刻，那么其必然是在 $[t_1, t_2]$ 中的某一时刻退出了直播间。更进

一步地，这个时间段的下界还可以根据有没有 action 变得更紧，即在 t_1 之后的某一时刻 t' 如果该用户有 action，那么退出直播间的时刻必定不小于 t'。下面假定不符合这些条件的事件概率均为 0，后面不再赘述这个事情。

现在假定不知道用户是在 t_2 时刻重新进入了直播间这一先验条件，可以对所有 $t \in [t_1, t_2]$ 计算 $P_i(t_1, t, t_2)$ 表示已知最后一次进入直播是在 t_1 时刻时，i 用户在 t 时刻退出直播间后又在 t_2 时刻重新进入的概率，它等于2.2和2.3中计算的两个概率的乘积，即 $P_i(t_1, t, t_2) = R_i(t_1, t) \cdot Q_i(t, t_2)$。据此，可以得到两个核心结论。

结论一：用户在 t_2 时刻重新进入了直播间的概率 $P'_i(t_1, t_2) = \sum_{t \in [t_1, t_2]} P_i(t_1, t, t_2)$。

结论二：当已知用户在 t_2 时刻重新进入了直播间的概率时，可以用贝叶斯公式计算其在 $t \in [t_1, t_2]$ 时刻退出直播间的概率 $\frac{P_i(t_1,t,t_2)}{P'_i(t_1,t_2)}$，这意味着该用户最有可能在 $\arg\max_{t \in [t_1, t_2]} P_i(t_1, t, t_2)$ 时刻退出直播间。

3 算法

3.1 参数化

我们把一个用户在整场直播中 action 的次数和金额总数视为其特征，认为函数 α_i 和 σ_i 的计算以及参数 λ_i 和 β_i 与其相关，那么有 $\alpha_i(c) = \alpha(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j, c)$，$\sigma_i(c) = \sigma(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j, c)$，$\lambda_i = \lambda(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j)$，$\beta_i = \beta(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j)$。

可以为上述函数设定可调节的参数。在此为简便起见，设置参数如下：

$$\alpha(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j, c) = C_i + \sum_{1 \le j \le C_i} \mathbf{C_i}_j + c + 1$$

$$\beta(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j) = C_i + \sum_{1 \le j \le C_i} \mathbf{C_i}_j + 1$$

$$\lambda(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j) = (\theta_0 C_i + \theta_1 \sum_{1 \le i \le C_i} \mathbf{C_i}_j + \theta_2)^{-1}$$

$$\sigma(C_i, \sum_{1 \le j \le C_i} \mathbf{c_i}_j, c) = \theta_3 C_i + \theta_4 \sum_{1 \le j \le C_i} \mathbf{C_i}_j + \theta_5 c + \theta_6$$

我们认为，如果用户行为越活跃（参数越大），则对应热情曲线衰减越慢，依此设置了相应的单调性关系。

如果确定了 θ，就意味着以上的函数和参数都确定了；关于如何确定 θ，我们在3.2中讨论。

3.2 最大似然估计

目前 θ 是未知的。根据最大似然估计的思想，我们考虑找到一个 θ_0，在用其作为参数的情况下，当前数据提供信息的发生概率最大。我们用3.2中的 P' 作为这里讨论的概率，求解 $\arg\max_\theta \prod_{1 \le i \le n} \prod_{2 \le j \le E_i} P'(\mathbf{e_i}_{j-1}, \mathbf{e_i}_j) = \arg\max_\theta \sum_{1 \le i \le n} \sum_{2 \le j \le E_i} \log P'(\mathbf{e_i}_{j-1}, \mathbf{e_i}_j)$。

在求得 θ_0 之后，直接根据3.2中的结论二，对于所有 $1 \le i \le n, 1 \le j \le E_i - 1$ 计算得 $\widetilde{s_{ij}}$。

我们使用了梯度下降的方法求得了一个（局部）最优的 θ_0，把其作为我们的参数。最后，可以直接用中的结论二估计每个用户每次进入直播间后的停留时间。

3.3 最后一次进入直播间的退出时刻估计

此前的方法有一个特例情况，即不能直接适用于估计每个用户最后一次进入直播间后的退出时间。主要缘由有二，一是由于这是该用户最后一次进入直播间，我们不再有其下一次进入直播间的先验知识，二是该用户不一定中途退出，而是直接观看到比赛结束时刻 $T-1$ 。对于第二种情况，我们不妨记该观众的最后一次离开时刻 $s_{i,E_i} = T-1$ 。

由现实意义理解，越晚进入直播间的观众，由于比赛快要结束，其往往更倾向于看完比赛，而不是在很短的时间内离开。如，最后 3 分钟进入直播间的观众，更大概率会直接把这 3 分钟比赛看完。由此，我们可以对原有方法作一些修正来粗略估计这种情况下的离开时刻。

我们根据经验设置一个参数 δ ,并将模型中的比赛结束时间延长至 $T+\delta$ (显然,$[T, T+\delta)$ 中不会有 action)，假设每个用户在最后一次进入直播间后，还在 $T+\delta$ 时刻重新进入直播间。我们以修正后的时刻上限计算此前 2.2 与 2.3 提出的重新进入直播间概率 Q_i 和退出直播间时刻概率 P_i 。用贝叶斯公式计算用户 i 在 $t \in [e_{i,E_i}, T+\delta)$ 时刻的退出概率，并找到在该模型下算出的最有可能的退出时刻 $\widetilde{s_{0i,E_i}}$。

分两种情况讨论：

(1) 若 $\widetilde{s_{0i,E_i}} < T-1$ ，则 $\widetilde{s_{i,E_i}} = \widetilde{s_{0i,E_i}}$ ，此时意味着用户 i 在比赛结束前中途离开直播间。

(2) 若 $\widetilde{s_{0i,E_i}} \ge T-1$ ，则 $\widetilde{s_{i,E_i}} = T-1$ ，此时意味着用户 i 一直观看到了比赛结束。

A 代码

我们在此处给出我们的实现代码。我们使用的语言是 python，版本为 3.9.7。

```
from turtle import forward
from openpyxl import load_workbook, worksheet
from requests import get
import torch
from torch import nn
import numpy as np
import json
from datetime import datetime
from tqdm import tqdm

wb = load_workbook("A题数据.xlsm", data_only=True)
sheets = [None]
for i in range(1, 5):
    sheets.append(wb[f"Sheet{i}"])
```

```
for i, sheet in enumerate(sheets):
    if i == 0:
        continue
    with open(f"sheet{i}", "w") as file:
        for letter in "ABCDEFG":
            file.write(
                letter + ":" + "!".join([str(x.value) for x in sheet[letter]]) + "\n"
            )

S = int(datetime.timestamp(datetime(2019, 12, 15, 20, 30, 0)))
T = int(datetime.timestamp(datetime(2019, 12, 15, 21, 20, 0)))


def get_column(sheet, cols, lmbda=lambda x: x):
    ret = []
    with open(f"sheet{sheet}", "r") as file:
        for line in file:
            for col in cols:
                if line.startswith(col):
                    ret.append(
                        [lmbda(x) for x in line.split(col + ":")[1].strip().split("!")]
                    )
                    if sheet == 2:
                        ret[-1] = ret[-1][2:]
                    else:
                        ret[-1] = ret[-1][1:]
    return ret


danmu_time = get_column(3, "AB")
liwu_time = get_column(4, "AE")
liwu_cost = get_column(4, "AD")
valid_user_list = get_column(1, "A")[0]
valid_user = set()
for i in valid_user_list:
    valid_user.add(i)

ts_dict = {}
for user in valid_user_list:
    ts_dict[user] = []


def valid(ip):
    return ip in valid_user


for ip, time in zip(danmu_time[0], danmu_time[1]):
    if not valid(ip):
        continue
    time = datetime.timestamp(datetime.strptime(time, "%Y-%m-%d %H:%M:%S"))
    time = int(time)
    ts_dict[ip].append((time, 0))

for ip, time, cost in zip(liwu_time[0], liwu_time[1], liwu_cost[1]):
```

```
    if not valid(ip):
        continue
    time = datetime.timestamp(datetime.strptime(time, "%Y-%m-%d %H:%M:%S"))
    time = int(time)
    cost = float(cost)
    ts_dict[ip].append((time, cost))

ts = [
    np.array([y[0] for y in sorted(ts_dict[x], key=lambda t: t[0])])
    for x in sorted(ts_dict)
]
cs = [
    np.array([y[1] for y in sorted(ts_dict[x], key=lambda t: t[0])])
    for x in sorted(ts_dict)
]
cis = [len(x) for x in cs]
scis = [np.sum(x) for x in cs]

in_time = get_column(1, "AB")
tin_dict = {}
for user in valid_user_list:
    tin_dict[user] = []

for ip, time in zip(in_time[0], in_time[1]):
    time = datetime.timestamp(datetime.strptime(time, "%Y-%m-%d %H:%M:%S"))
    time = int(time)
    tin_dict[ip].append(time)

tin = [np.array(sorted(tin_dict[x])) for x in sorted(tin_dict)]

n = len(ts)

theta = [0.001, 0.001, 0.001, 20, 20, 1, 1]


def alpha(i, c):
    return cis[i] + scis[i] + c + 1


def beta(i):
    return cis[i] + scis[i] + 1


def sigma(i, c):
    return cis[i] * theta[3] + scis[i] * theta[4] + c * theta[5] + theta[6]


def lmbda(i):
    return 1 / (cis[i] * theta[0] + scis[i] * theta[1] + theta[2])


def fi(i, t, u, c):
    return np.exp(-((t - u) ** 2) / (2 * sigma(i, c) ** 2)) / (sigma(i, c))
```

```
pats = []
for i in tqdm(range(n)):
    pats.append([])
    for t in range(S, T):https://www.overleaf.com/project/6263d1b3bbaad2f1b10f0e35
        pats[-1].append(0)
        for j in range(len(ts[i])):
            if ts[i][j] <= t:
                pats[-1][-1] += alpha(i, cs[i][j]) * fi(i, t, ts[i][j], cs[i][j])

with open("pats.txt", "w") as file:
    for user in pats:
        file.write(" ".join([str(x) for x in user]) + "\n")

pats_obj = {}
with open("pats.txt", "r") as file:
    for i, line in tqdm(enumerate(file)):
        if cis[i] == 0:
            pats_obj[i] = None
        else:
            line = [float(x) for x in line.split()]
            pats_obj[i] = line

with open("pats.json", "w") as file:
    file.write(json.dumps(pats_obj))

with open("pats.json", "r") as file:
    pats_obj = json.loads(file.read())

sumpat = []

for t in tqdm(range(S, T)):
    sumpat.append(0)
    for i in range(n):
        sumpat[-1] += pats[i][t - S]

with open("sumpat.txt", "w") as file:
    file.write(" ".join([str(x) for x in sumpat]) + "\n")
sumpat_obj = {}
with open("sumpat.txt", "r") as file:
    sumpat = [float(x) for x in file.read().split()]
    for i, j in enumerate(sumpat):
        sumpat_obj[i] = j
with open("sumpat.json", "w") as file:
    file.write(json.dumps(sumpat_obj))

with open("sumpat.json", "r") as file:
    sumpat_obj = json.loads(file.read())


sumR = []

for i in tqdm(range(n)):
    betai = beta(i)
    sumR.append([0 for _ in range(T - S + 1)])
    # if cis[i] == 0:
```

```
    #     continue
    for t in range(T - 1, S - 1, -1):
        sumR[-1][t - S] = (
            sumR[-1][t + 1 - S] + 1 / (1 + pats[i][t - S]) + betai / (1 + sumpat[t - S]
                                                                      )
        )

with open("sumR.txt", "w") as file:
    for user in sumR:
        file.write(" ".join([str(x) for x in user]) + "\n")

sumR_obj = {}
with open("sumR.txt", "r") as file:
    for i, line in tqdm(enumerate(file)):
        if cis[i] == 0:
            sumR_obj[i] = None
        else:
            line = [float(x) for x in line.split()]
            sumR_obj[i] = line

with open("sumR.json", "w") as file:
    file.write(json.dumps(sumR_obj))

sumRpat = [0] * (T - S)
with open("sumR.json", "r") as file:
    sumR_obj = json.loads(file.read())


def pat_f(i, t):
    i = str(i)
    if pats_obj[i] is None:
        return 0
    return pats_obj[i][t - S]


def sumpat_f(t):
    return sumpat_obj[str(t - S)]


def sumR_f(i, t):
    betai = beta(i)
    i = str(i)
    if sumR_obj[i] is None:
        sum = 0
        if sumRpat[t - S] == 0:
            for tt in range(t, T):
                sum += 1 / (1 + sumpat_f(tt))
            sumRpat[t - S] = sum
        else:
            sum = sumRpat[t - S]
        return (T - 1 - t) + betai * sum
    return sumR_obj[i][t - S]


def Q(i, t1, t):
```

```
    li = lmbda(i)
    up = np.exp(-li * (t - t1))
    down = (1 - np.exp(-li * (T - t1 - 1))) / (1 - np.exp(-li))
    return up / down


def R(i, t1, t):
    betai = beta(i)
    up = 1 / (1 + pat_f(i, t)) + betai / (1 + sumpat_f(t))
    down = sumR_f(i, t1)
    return up / down


def P(i, t1, t2):
    sum = 0
    for t in range(t1, t2 + 1):
        sum += Q(i, t1, t) * R(i, t, t2)
    return sum


def lossi(i):
    if cis[i] < 2:
        return 0
    loss = 0
    for j, _ in enumerate(tin[i]):
        if j == 0:
            continue
        if tin[i][j] >= T:
            continue
        loss -= np.log(P(i, tin[i][j - 1], tin[i][j]))
    return loss


def loss():
    sum = 0
    for i in tqdm(range(n)):
        sum += lossi(i)
    return sum / n


print(1)
print(loss())
```

B 最优参数

我们得到的最优参数为

$$\theta = [0.001, 0.001, 0.001, 0.785, 0.767, 1, 1]$$