Optimal Dose for mHealth Interventions

Yongjun Lee

Introduction

Mobile health (mHealth) refers to the use of mobile devices, such as smartphones and wearable devices, to deliver healthcare services and interventions, and also detect the current health status of the individual. These interventions are usually intended to promote healthy behavioral change. For example, we may consider sending smartphone push notification for medication reminders or physical activity suggestions for sedentary workers. The efficacy of these interventions are analyzed through microrandomized studies where participants are randomized to treatment (e.g. push notification) multiple times throughout the day, and outcome is also recorded through a smart device (e.g. step count).

In the mHealth literature, previous research has examined the delayed effects of notifications or the immediate treatment effect within a 30-minute window following an intervention. However, focusing on a single treatment effect may not be reliable since too few number of treatments might not be sufficient for the desired outcome, while an excessive number of treatments could lead to participant fatigue, potentially resulting in adverse effects. To address this issue, I am in the process of developing a method to efficiently analyze the optimal "dose" of daily treatment.

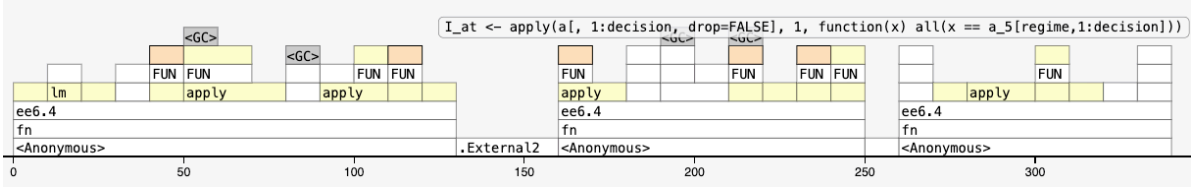
The estimand of interest is the dose effect which is defined by $\mathbb{E}[Y(dose) - Y(0)|S] = S^\top \beta$, where $Y(dose)$ is the outcome under treatment d, $Y(0)$ is the outcome under no treatment, and S is the vector of features that affect the outcome. It can be shown that the dose effect can be estimated by solving the following estimating equation.

$$\tilde{u} = \sum_{t=1}^{T} \sum_{k=1}^{K} \frac{1}{\binom{k}{d}} \sum_{\bar{a}_k \in \mathcal{D}} \left(\frac{\mathbb{1}(\bar{A}_k = \bar{a}_k)}{\mathbb{P}(\bar{A}_k = \bar{a}_k)} \left(Y_{t,k} - m_1\right) - \frac{\mathbb{1}(\bar{A}_k = \bar{0}_k)}{\mathbb{P}(\bar{A}_k = \bar{0}_k)} \left(Y_{t,k} - m_0\right) + m_1 - m_2 - \frac{1}{K} S^\top \beta \right) \nu(S)$$

Here, T is total number of follow-up days, K is the number of randomization time points within a day, $\bar{a}_k$ is the treatment regime up to time k, $\mathcal{D}$ is the set of all treatment regimes for dose d, $\bar{A}_k$ is the observed treatment vector for day t, up to decision point k, $Y_{t,k}$ is the outcome on day t at decision point k, m_1 is the predicted outcome regression model of the outcome with baseline covariates had everyone received treatment k, and m_0 is the predicted

outcome from same regression model under no treatment, S is the set of pre-randomization effect modifying variables, and β is the dose effect.

Code Improvement 1 - Vectorization



The above profvis output shows that the bottleneck of the code is for computing $\mathbb{1}(\bar{A}_k = \bar{a}_k)$ and $\mathbb{1}(\bar{A}_k = \bar{0}_k)$. Thse two indicators identify the individuals who received the treatment regime $\bar{a}_k$ and those who received no treatment until decision point k. The current implementation uses `apply` function that loops to iterate over each individual and check if the treatment regime is equal to $\bar{a}_t$ or $\bar{0}_t$. Also, it creates `rep(0,decision)` for each iteration which adds to memory allocation and deallocation overhead.

```
I_at <- apply(a[, 1:decision, drop=FALSE], 1,
function(x) all(x == a_5[regime,1:decision]))
I_0t <- apply(a[, 1:decision, drop=FALSE], 1,
function(x) all(x == rep(0, decision)))
```

This is inefficient since the code is not vectorized and `all()` function is run for every iteration. To improve the code, I vectorized the code so that it check the entire matrix at once. Also, I predefined `zeros` vectors to reduce overhead cost. The new code snippet and the resulting benchmark is shown below. The new code is almost 3 times faster than the original code.

```
I_at <- rowSums(a[, 1:decision, drop=FALSE] == matrix(a_5[regime, 1:decision],
nrow=nrow(a), ncol=decision, byrow=TRUE)) == decision
I_0t <- rowSums(a[, 1:decision, drop=FALSE] == matrix(zeros[[decision]],
nrow=nrow(a), ncol=decision, byrow=TRUE)) == decision
```

func	min	median	itr/sec	n_itr	total_time
Original	2.813034	2.813366	1.000000	11	503ms
Improved	1.000000	1.000000	2.626648	29	505ms

Code Improvement 2 - Parallel Computing

To estimate the variance of our estimator, we use bootstrapping method where multiple datasets are generated and we calculate the empirical standard error of the estimated value for each dataset. Since each bootstrapping iteration is not dependent of each other, this is optimal for applying the parallelization technique. This can be easily implemented by replacing the `for` loop with `mclapply` function from `parallel` package. The resulting benchmark is shown below. The parallelized code is also two times faster than the original code.

func	min	median	itr/sec	n_itr	total_time
Original	2.650129	2.623218	1.000000	1	1.25s
Improved	1.000000	1.000000	2.623218	2	954.15ms

Simlations

Some simulations were carried out to verify the consistency of our method and also to check the computaion time for differing sample size and simulation iterations. The data generating process is:

$$Y_t = H\eta + S\beta a_t + \epsilon$$

where $H = (\bar{h}_1, \bar{h}_2, \bar{h}_3), h_1 = \bar{1}_n, h_2 \sim Normal(0,1), h_3 = h_2^2, \eta = (30,20,10), S = (s_1, s_2, s_3), s_1 = \bar{1}_n, s_2 \sim Binom(0.5), s_3 \sim Normal(0,1), \beta = (20,10,5), \epsilon \sim Normal(0,1)$ and a_t is the treatment regime at time t. The data is generated for $n = 50, 100$ individuals, $t = 10, 20$ days, $m = 100, 200$ simulations. The dose effect is $\beta = (20, 10, 5)$ and the dose is 1. The resulting simulation results show consistency, and the computation time is linear with respect to the number of individuals and days.

	n	days	m	beta1	hat1	sd1	runtime
rslt1	50	10	100	20	20.39779	9.769794	3.900814
rslt2	100	20	100	20	19.83706	4.054526	6.960800
rslt3	100	20	200	20	20.24290	4.128236	13.545952