Optimizing User Retention in Cookie Cats: A Bayesian A/B Testing and Thompson Sampling Approach

Kamala Dadashova

Cookie Cats is a mobile puzzle app. We analyze the Cookie Cats data available at https://www.kaggle.com/datasets/yufengsui/mobile-games-ab-testing. The data were collected by the developers to improve its appeal. The app was deployed to users with the level of the first gate set to either 30 or 40, and the response was one-day retention of the user. Level 30 was given to $n_1 = 44700$ users and $Y_1 = 20034$ were retained; level 40 was given to $n_2 = 45489$ users and $Y_2 = 20119$ were retained.

1. A/B Testing

Let θ_1 and θ_2 be the true retention probabilities under the two levels. A/B testing is often used to find the optimal setting for an app or website. The final output used for decision making is the posterior probability that each setting is optimal. For Cookie Cats, we say level 30 is optimal if $\theta_1 > \theta_2$ and level 40 is optimal if $\theta_2 > \theta_1$.

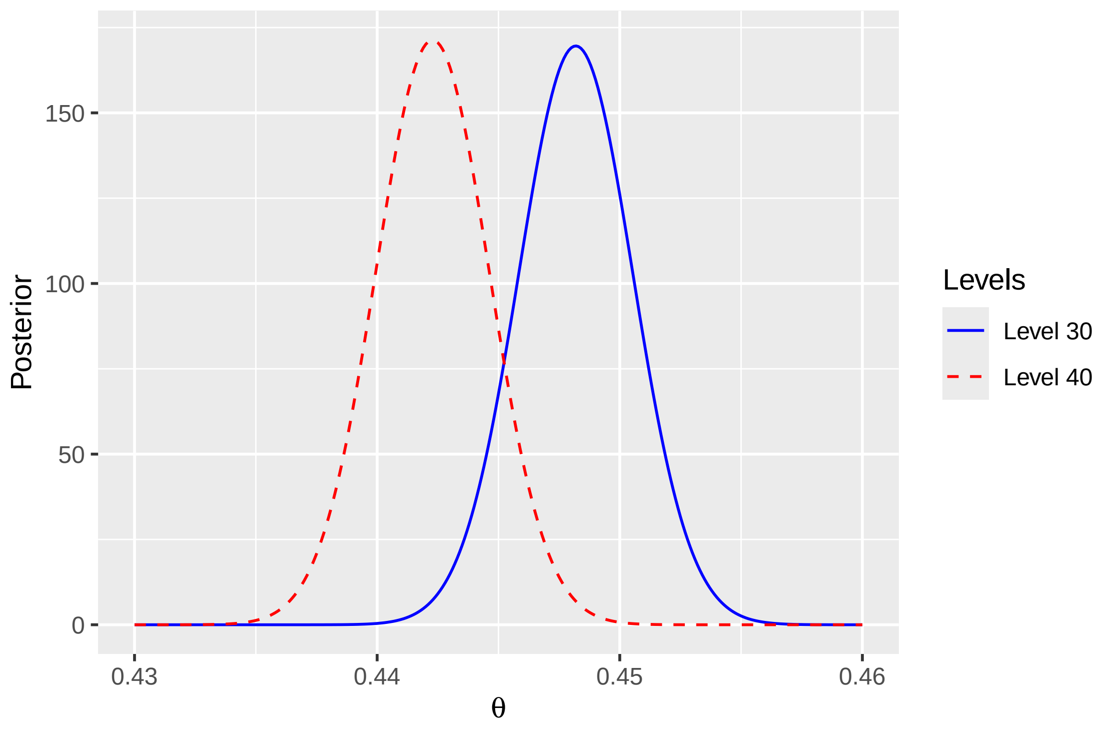
(A) Let's first determine the posterior distributions of θ_1 and θ_2. The responses are the number of successes in a fixed number of trials; therefore, the likelihood $Y_j|\theta_j \sim \text{Binomial}(n_j, \theta_j)$ and conjugate prior $\theta_j \sim \text{Beta}(a, b)$, independent for $j = 1, 2$.Settting $a = b = 1$ give an uninformative prior. The posteriors are $\theta_j|Y_j \sim \text{Beta}(Y_j + a, n_j - Y_j + b)$.

(B) The plot for the posterior distributions of θ_1 and θ_2 is.

```r
# Load necessary libraries
suppressPackageStartupMessages({
  library(tibble)
  library(ggplot2)
  library(tidyr)
  library(dplyr)
})
```

```r
# Define constants and initial values
n1=44700; Y1=20034; n2=45489; Y2=20119; a=1; b= 1

# Generate theta sequence
theta = seq(0.43, 0.46, 0.0001)

# Compute beta density values for each theta
theta1 = dbeta(theta, Y1 + a, n1 - Y1 + b)
theta2 = dbeta(theta,  Y2 + a,  n2 - Y2 + b)

# Create data frame for plotting
df = data.frame(theta, theta1, theta2)
```

```r
# Rename columns for clarity
colnames(df) = c("theta", "theta1", "theta2")

# Transform data frame for ggplot
# Transform data frame for ggplot
df <- df %>%
  pivot_longer(cols = c("theta1", "theta2"), names_to = "Levels", values_to = "Posterior")

# Plot the beta distributions with custom legend labels
plt <- ggplot(df, aes(x = theta, y = Posterior, color = Levels, linetype = Levels)) +
  geom_line() +
  theme(plot.margin = unit(c(1, 1, 1, 1), "cm")) +
  labs(x = expression(theta)) +
  scale_color_manual(labels = c("Level 30", "Level 40"), values = c("blue", "red")) +
  scale_linetype_manual(labels = c("Level 30", "Level 40"), values = c("solid", "dashed"))

plot(plt)
```



Based on this plot, the posterior mean retention rate is marginally higher for Level 30. Furthermore, posteriors for Level 30 and Level 40 overlap.

(C) To determine the posterior probability that each level is optimal, I use MCMC sample to approximate them.

```
theta1 =rbeta(1000000,Y1+a,n1-Y1+b)
theta2 =rbeta(1000000,Y2+a,n2-Y2+b)
mean(theta1>theta2)
```

```
## [1] 0.9624
```

Based on this result,

$$P(\theta_1 > \theta_2 \mid Y_1, Y_2) = 0.96$$

and

$$P(\theta_2 > \theta_1 \mid Y_1, Y_2) = 0.04$$

. Therefore, Level 30 is more likely to be optimal.

(D) In order to check the sensitivity of results obtained above, I conduct a prior sensitivity analysis.

```
ab_vals= c(0.01,0.1,1,10,100) ;  p=ab_vals; options(scipen = 999)
for(j in 1:length(ab_vals)){ theta1 =rbeta(1000000,Y1+ab_vals[j],n1-Y1+ab_vals[j])
theta2=rbeta(1000000,Y2+ab_vals[j],n2-Y2+ab_vals[j])
p[j]  <-mean(theta1>theta2) }
cbind(ab_vals,p)
```

```
##      ab_vals        p
## [1,]    0.01 0.963031
## [2,]    0.10 0.962740
## [3,]    1.00 0.962845
## [4,]   10.00 0.962455
## [5,]  100.00 0.962443
```

There is no sensitivity to the prior as I change the values of a and b.

2. Bayesian Bandit

Instead of a static A/B testing approach, one can implement a dynamic approach where data is collected and analyzed sequentially. Each day, 100 users are sampled, and based on the results of the previous days, a decision is made on which level (30 or 40) to assign to these users.

Day (t)	1	2	3	4
Level	30	40	40	30
Successes	30	50	40	20
n_1	100	100	100	200
Y_1	30	30	30	50
n_2	0	100	200	200
Y_2	0	50	90	90

Day(t): This column represents the day number in the sequential experiment. For example, $t = 1$ refers to the first day, $t = 2$ to the second day, and so on.

Level: This column indicates which level (30 or 40) was assigned to the users on that particular day. For example, on Day1, Level 30 was assigned, and on Day 2, Level 40 was assigned.

Successes: This column shows the number of users retained (i.e., who continued using the app) out of the 100 users sampled on that day. For example, on Day1, 30 users were retained out of 100, and on Day2, 50 users were retained out of 100.

$n1$: This column represents the cumulative number of users assigned to Level 30 up to that day. For example, on Day 1, 100 users were assigned to Level 30 ($n_1 = 100$), and by Day 4, a total of 200 users were assigned to Level 30 ($n_1 = 200$).

$Y1$: This column shows the cumulative number of successes (retained users) for Level 30 up to that day. For example, by Day 1, there were 30 retained users for Level 30 ($Y1 = 30$), and by Day 4, there were 50 retained users for Level 30 ($Y1 = 50$).

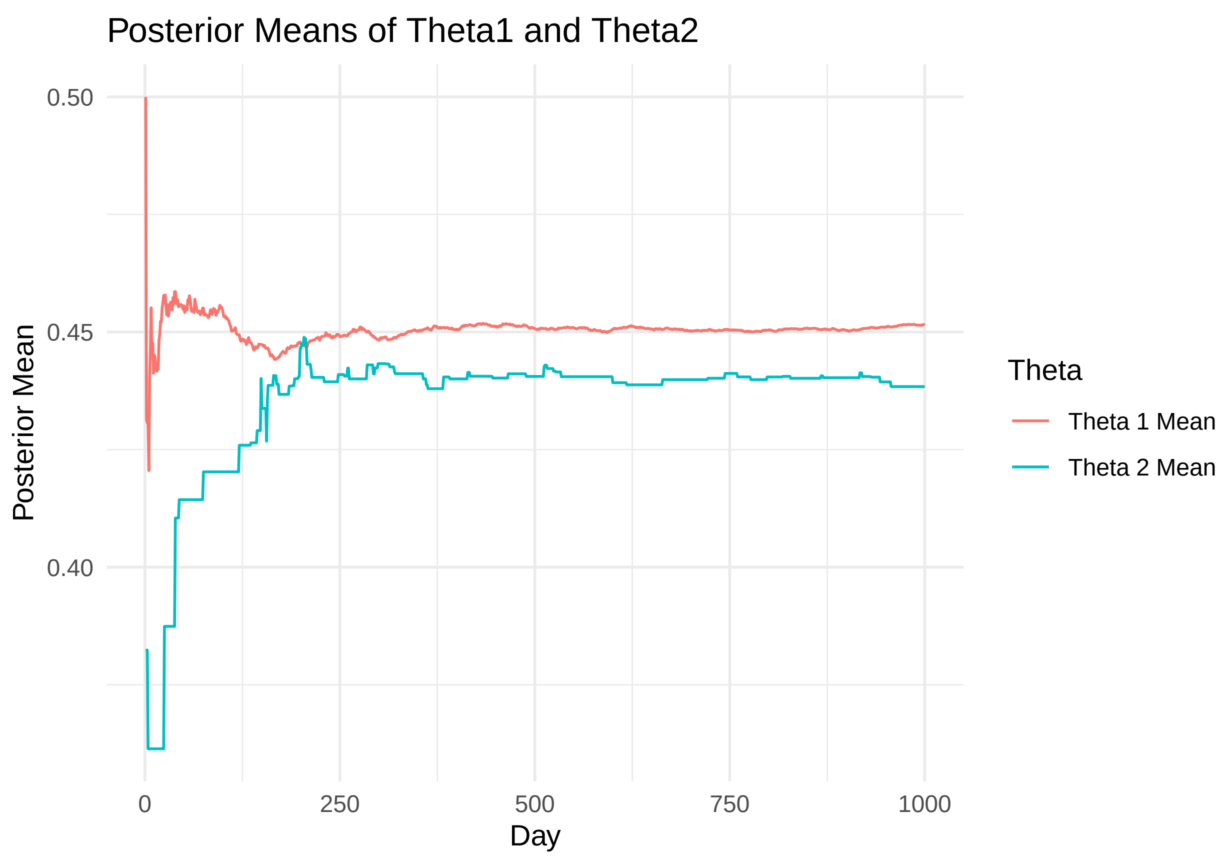
I use Thompson sampling to decide which treatment to give (to all 100 users) each day. On day t, you will draw one sample of θ_1 and one sample of θ_2, denoted θ_1^* and θ_2^*, from their posteriors based on all data collected prior to day t, and if $\theta_1^* > \theta_2^*$ you will give level 30, otherwise you will give level 40. Of course, we cannot get real data, so instead I take a subsample (with replacement) of the original data, i.e., if you decide to give level 30 then the data for that day are simulated as $Y \sim \text{Binomial}(1, 100, \frac{20034}{44700})$.

(A) To implement this sampling scheme, I provide a function that takes the prior hyperparameters and current data as inputs and gives the random treatment assignment as output.

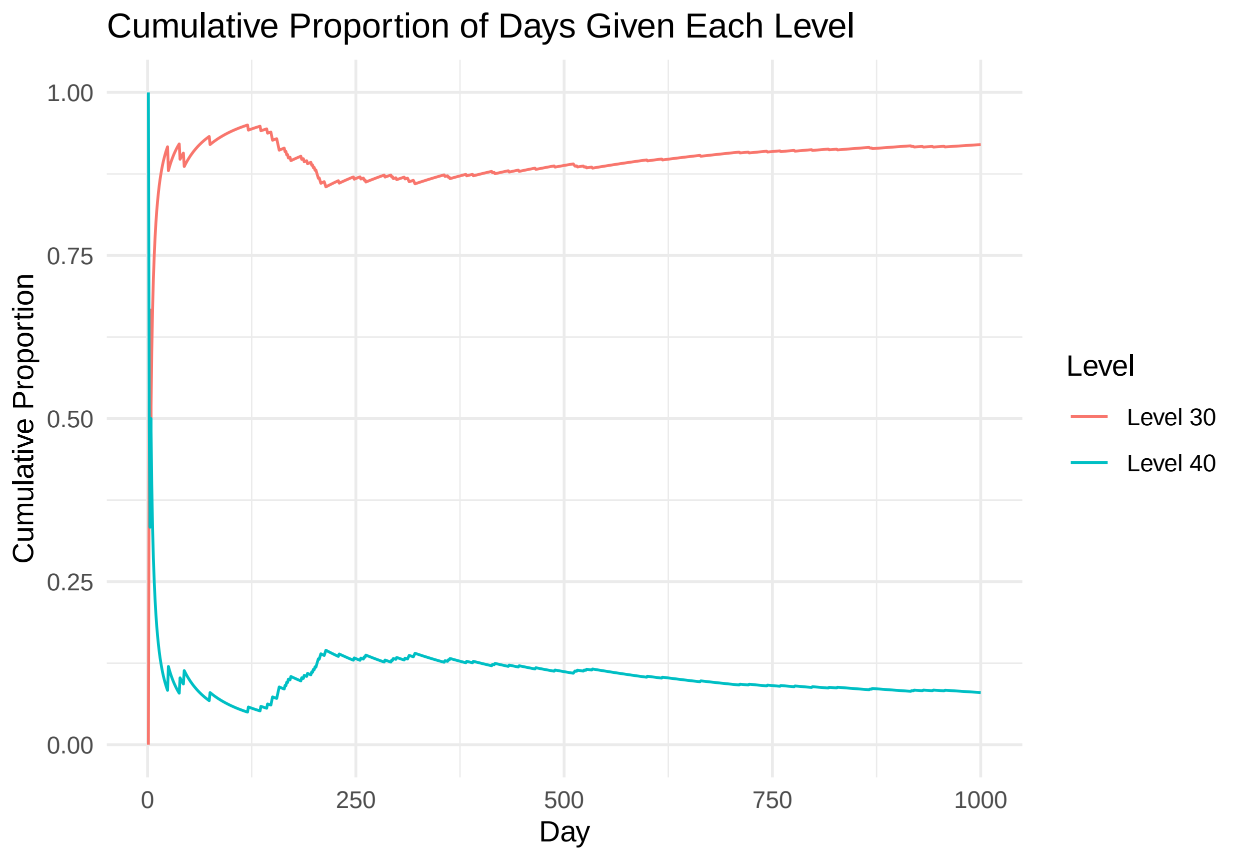
```r
# Likelihood: Y_j ~ Binom(n_j,theta_j) j=1,2
# Prior: theta_j~Beta(a,b)
# Inputs: Y1,Y2,n1,n2,a,b
# Output: group to be sample, i.e.,  1 or 2
# Thompson Sampling Function
Thompson_sampling = function(Y1, n1, Y2, n2, a = 1, b = 1) {
  theta1_sample = rbeta(1, a + Y1, b + n1 - Y1)
  theta2_sample = rbeta(1, a + Y2, b + n2 - Y2)
  if (theta1_sample > theta2_sample) {
    return(1) # Level 30
  } else {
    return(2) # Level 40
  }
}
```

(B) I carry out this algorithm for 1000 days, which is followed by plotting the cumulative proportion of days given each level and the posterior means of θ_1 and θ_2 as a function of t.

```r
# Load necessary libraries
library(ggplot2)
library(dplyr)

# Initial parameters
days = 1000
p =c(20034 / 44700, 20119 / 45489) # True probabilities for levels 30 and 40
a=b=1 # Prior hyperparameters

# Initialize matrices to store results
n =Y =matrix(0, days + 1, 2)

# Simulation loop for 1000 days
for (t in 1:days) {
  set.seed(919 * t) # Set seed for reproducibility
  # Thompson Sampling to pick the level
  level= Thompson_sampling(Y[t, 1], n[t, 1], Y[t, 2], n[t, 2], a, b)

  # Generate new data based on selected level
  y = rbinom(1, 100, p[level])

  # Update counts for the selected level
  Y[t + 1, ] = Y[t, ]
  n[t + 1, ] = n[t, ]

  if (level == 1) {
    Y[t + 1, 1] = Y[t, 1] + y
    n[t + 1, 1] = n[t, 1] + 100
  } else {
    Y[t + 1, 2] = Y[t, 2] + y
    n[t + 1, 2] = n[t, 2] + 100
  }
}

# Remove initial row (time 0)
Y = Y[-1, ]
n = n[-1, ]

# Calculate posterior means
posterior_mean= (Y + a) / (n + a + b)

# Calculate cumulative sampling proportions
cum_n = cbind(n[, 1] / (n[, 1] + n[, 2]), n[, 2] / (n[, 1] + n[, 2]))

# Prepare data for plotting
results = data.frame(
  day = 1:days,
  theta1_mean = posterior_mean[, 1],
  theta2_mean = posterior_mean[, 2],
  cum_prop_30 = cum_n[, 1],
  cum_prop_40 = cum_n[, 2]
)

# Plot posterior means of theta1 and theta2
ggplot(results, aes(x = day)) +
  geom_line(aes(y = theta1_mean, color = "Theta 1 Mean")) +
  geom_line(aes(y = theta2_mean, color = "Theta 2 Mean")) +
  labs(title = "Posterior Means of Theta1 and Theta2",
       x = "Day", y = "Posterior Mean", color = "Theta") +
  theme_minimal()
```



```r
# Plot cumulative sampling proportions
ggplot(results, aes(x = day)) +
  geom_line(aes(y = cum_prop_30, color = "Level 30")) +
  geom_line(aes(y = cum_prop_40, color = "Level 40")) +
  labs(title = "Cumulative Proportion of Days Given Each Level",
       x = "Day", y = "Cumulative Proportion", color = "Level") +
  theme_minimal()
```



Result: θ_1 has a higher posterior mean compared to θ_2, suggesting that level 30 has a higher retention rate. In the initial phase, the cumulative proportion fluctuates as the algorithm explores both levels. Level 30 (red line) is chosen much more frequently compared to level 40 (blue line). The red line (level 30) converges to a high cumulative proportion, close to 1, indicating that level 30 is chosen almost all the time. The blue line (level 40) converges to a low cumulative proportion, close to 0, indicating that level 40 is rarely chosen.

(C) What is difference (pros ans cons) of using Thompson sampling versus single experiment in Problem 1?

Advantages of Thompson Sampling

As shown in 2b, an advantage of Thompson sampling explores both levels early when the optimal level is uncertain, but in the long-run most users are given the preferred level.

- **Early Exploration:**
  - Thompson sampling initially allocates users to both levels (or treatments) to gather sufficient data about each level's performance. This exploration phase helps the algorithm to understand the underlying success probabilities of each level, especially when there is uncertainty about which level is better.
  - For example, imagine you are testing two versions of a mobile app. Early on, Thompson sampling will assign users to both versions to collect data on user retention rates. This way, it gets a sense of which version might be better.
- **Long-Term Exploitation:**

  - As more data is collected, Thompson sampling updates the posterior distributions of the success probabilities and increasingly favors the level that performs better. This ensures that over time, most users are allocated to the level with the higher probability of success, maximizing the overall performance (e.g., user retention).
  - For example, continuing with the mobile app example, once the algorithm has gathered enough data, it will primarily assign new users to the version with the higher retention rate, thereby improving overall user retention.

Disadvantages of Thompson Sampling

Disadvantages are that implementing Thompson sampling brings some logistical challenges and it is possible that the algorithm gets stuck on a sub-optimal level.

- **Logistical Challenges:**
  - Implementing Thompson sampling can be more complex than a simple A/B test. It requires continuous updating of the posterior distributions, random sampling from these distributions, and dynamic allocation of users based on the sampled values.
  - In a real-world scenario, maintaining the computational infrastructure to perform these continuous updates and allocations can be resource-intensive. It requires sophisticated software and statistical expertise.
- **Risk of Sub-Optimal Convergence:**
  - There is a risk that Thompson sampling might get stuck on a sub-optimal level, especially if early data is misleading or highly variable. If the early samples favor a sub-optimal level due to random chance, the algorithm might continue to allocate more users to this level, reinforcing the initial incorrect belief.
  - Suppose the initial users randomly allocated to version A of the app have unusually high retention due to external factors (e.g., a marketing campaign). The algorithm might wrongly conclude that version A is better and continue to favor it, even if version B is actually superior in the long run.

Summaru of both approaches

- **Adaptability:**
  - **Thompson Sampling:** Continuously adapts and reallocates based on performance, improving over time.
  - **A/B Test:** Fixed allocation, no adaptation during the experiment.
- **Complexity:**
  - **Thompson Sampling:** More complex to implement and requires continuous updating.
  - **A/B Test:** Simpler to implement and analyze.
- **Performance:**
  - **Thompson Sampling:** Potentially higher overall performance due to adaptive reallocation.
  - **A/B Test:** Potentially lower overall performance as it does not adapt.
- **Use of Data:**
  - **Thompson Sampling:** Efficient use of data, balancing exploration and exploitation.
  - **A/B Test:** Inefficient use of data if one level is significantly better.