```
---
title: "Bernoulli Thompson Sampling 1"
output: html_document
date: "2024-10-02"
---

```{r setup, include=FALSE}

# Packes required for subsequent analysis. P_load ensures these will be installed and
loaded.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse,
               ggplot2,
               devtools,
               BiocManager,
               contextual
               )

knitr::opts_chunk$set(echo = TRUE, eval = FALSE)

```

#Dataset

To manage the large dataset more efficiently, we grouped the data by arm and sampled 4%
from each group. This approach reduces the computational load while preserving the
representativeness of the original dataset. Additionally, we shuffled the sampled data
to eliminate any potential ordering bias, because in the original dataset before any
transformation the dataset is not ordered by item_id but when we group it, it becomes
ordered.

```{r}
# this version contains 80 arms (ZOZO)
set.seed(0)
library(readr)
dfZozo_80 <- read_csv("~/Downloads/zozo_Context_80items.csv.zip")%>%
  select(item_id, click) %>%
  group_by(item_id) %>%
  sample_frac(0.04) %>%
  ungroup()%>%          # Ungroup to remove the group structure
  arrange(sample(n()))   # Randomly shuffle the rows


```
```
# Thompson Sampling: notation

```{=tex}
\begin{enumerate}
  \item The reward $Q_t(a)$ for each arm follows a beta distribution. At the start, at
$t = 0$, every arm is assumed to follow the same beta distribution with parameters
$\alpha_0 = 1$ and $\beta_0 = 1$. This means we initially assume each arm has an equal
probability of giving a reward.
  \item At each time $t$, the algorithm samples $n_{\mathrm{sample}}$ observations from
each of the distributions. The arm which has the highest average reward according to
the sample is the chosen arm.
  \item We then observe the reward $r_t$, and update the parameters of the distribution
accordingly.
  \begin{itemize}
    \item $\alpha_t = \alpha_{t-1} + r_t$
    \item $\beta_t = \beta_{t-1} + 1 - r_t$
  \end{itemize}
\end{enumerate}
```

\textbf{Task 1: select from the dataframe below an arm according to the Thompson
Sampling algorithm}. Per arm, the alpha and beta parameters have been given. Use
$n_{\mathrm{sample}}=100$.

```{r}

# set the seed
set.seed(0)

# arm index
arm <- 1:80

# alpha per arm
alpha <- rep(1, length(arm))

# beta per arm
beta <- rep(1, length(arm))

# dataframe with alpha, beta per arm
df_alpha_beta <- data.frame(arm=arm,alpha = alpha, beta = beta)

# number of samples to draw
n_sample <- 100

# create dataframe with sampled values
df_sampled <- mapply(rbeta, n_sample, df_alpha_beta$alpha, df_alpha_beta$beta)

# average per sample
avg_from_sampled <- apply(df_sampled, 2, mean)


# get the arm with the highest average
chosen_arm <- which.max(avg_from_sampled)

chosen_arm
```



```{r, results='hide', message=FALSE, warning=FALSE}

library(contextual)

# set the seed
set.seed(0)

## OfflineReplayEvaluatorBandit: simulates a bandit based on provided data
#
# Arguments:
#
#   data: dataframe that contains variables with (1) the reward and (2) the arms
#
#   formula: should consist of variable names in the dataframe. General structure is:
#       reward variable name ~ arm variable name
#
#   randomize: whether or not the bandit should receive the data in random order,
#              or as ordered in the dataframe.
#

# in our case, create a bandit for the data with 10 arms,
#   formula = click ~ item_id,
#   no randomization

bandit_Zozo_80 <- OfflineReplayEvaluatorBandit$new(formula = click ~ item_id,
                                                   data = dfZozo_80,
                                                   randomize = FALSE)

# lets generate 10 simulations, each of size 54266
size_sim <- 54266
n_sim<- 10
# here we define the Thompson Sampling policy object
```

```
TS          <- ThompsonSamplingPolicy$new()

# the contextual package works with 'agent' objects - which consist of a policy and a
bandit
agent_TS_zozo_80 <-  Agent$new(TS, # add policy
                               bandit_Zozo_80) # add bandit


## simulator: simulates a bandit + policy based on the provided data and parameters
#
# Arguments:
#
#   agent: the agent object, previously defined
#
#   horizon: how many observations from dataset used in the simulation
#
#   do_parallel: if TRUE, runs simulation in parallel
#
#   simulations: how many simulations?
#

simulator          <- Simulator$new(agent_TS_zozo_80, # set our agent
                                    horizon= size_sim, # set the size of each
simulation
                                    do_parallel = TRUE, # run in parallel for speed
                                    simulations = n_sim, # simulate it n_sim times
                                )


# run the simulator object
history_TS_zozo_80            <- simulator$run()

# gather results
df_TS_zozo_80 <- history_TS_zozo_80$data %>%
  select(t, sim, choice, reward, agent)



```

Now that we have gathered the results, let's dive a little bit deeper into each component. First, the bandit object, *OfflineReplayEvaluatorBandit*. The idea behind this function is that it 'replays' the results of an algorithm/policy based on random data. Once an algorithm picks an arm, for instance arm 1, it takes the first observation of arm 1 in the dataset. If the algorithm again picks arm 1, it again samples the next observation from arm 1, etc.

Let us calculate per simulation (1-10), the maximum number of observations.

```{r}
df_TS_zozo_80_max_t <- df_TS_zozo_80%>%
  group_by(sim) %>% # group by per agent
  summarize(max_t = max(t)) # get max t

df_TS_zozo_80_max_t
```
The results show that the maximum number of rounds across 10 simulations varied between 435 and 490, with simulation 8 reaching the highest number of rounds (490) and simulation 5 the lowest (435). While most simulations fall within a narrow range, this variation suggests differences in exploration patterns, where some simulations explored more arms or took more actions before reaching the end of the dataset.

In the dataframe *df_TS_zozo* we have gathered the results of the TS policy. This dataframe contains the following columns:

```{=tex}
\begin{itemize}
\item \textbf{$t$}: the step at which a choice was made
```

```
\item \textbf{sim}: the simulation for which we observe results
\item \textbf{choice}: the choice made by the algorithm
\item \textbf{reward}: the reward observed by the algorithm
\item \textbf{agent}: column containing the name of the agent
\end{itemize}
```

\textbf{Task 3: using this dataframe, make a plot of the average cumulative rewards for all simulations, together with the 95\% confidence interval}.

```{r}

# Max of observations. Adjusted to the number of observations per simulation (min 831) and the size of the dataset used
max_obs <- 650
# dataframe transformation
df_history_agg <- df_TS_zozo_80 %>%
  group_by( sim) %>% # group by simulation
  mutate(cumulative_reward = cumsum(reward))%>% # calculate, per sim, cumulative reward over time
  group_by( t) %>% # group by time
  summarise(avg_cumulative_reward = mean(cumulative_reward), # average cumulative reward
            se_cumulative_reward = sd(cumulative_reward, na.rm=TRUE)/sqrt(n_sim)) %>% # SE + Confidence interval
  mutate(cumulative_reward_lower_CI =avg_cumulative_reward - 1.96*se_cumulative_reward,
         cumulative_reward_upper_CI =avg_cumulative_reward + 1.96*se_cumulative_reward)%>%
  filter(t <=max_obs)


# TODO: Make the following two plots:
# 1: A plot that shows only the average cumulative rewards over time using the df_history_agg dataframe
# 2: The plot as defined in (1) together with the 95\% confidence interval.

legend <- c("Avg." = "orange", "95% CI" = "gray") # set legend


ggplot(data=df_history_agg, aes(x=t, y=avg_cumulative_reward)) +
  geom_line(size=1.5,aes(color="Avg.")) + # add line
  geom_ribbon(aes(ymin=ifelse(cumulative_reward_lower_CI<0, 0,cumulative_reward_lower_CI), # add confidence interval
                  ymax=cumulative_reward_upper_CI,
                  color = "95% CI"),
              alpha=0.1) +
  labs(x = 'Time', y='Average Cumulative Reward', color='Metric') + # add titles
  scale_color_manual(values=legend) + # add legend
  theme_bw() # set the theme

results <- data.frame(Round = numeric(), ChosenArm = numeric(), Reward = numeric()) # initialize results dataframe

for (i in 1:n_distinct(df_TS_zozo_80$sim)) { # ensure sim count is correct
  current_sim <- df_TS_zozo_80 %>% filter(sim == i)

  for (t in 1:nrow(current_sim)) {
    item_id <- current_sim$choice[t]
    click <- current_sim$reward[t]

    results <- rbind(results, data.frame(Round = t, ChosenArm = item_id, Reward = click))
  }
}

# Plot for Thompson Sampling Arm Choices Over Time
ggplot(results, aes(x = Round, y = as.factor(ChosenArm))) +
  geom_point(aes(color = as.factor(Reward))) +
```

```
  labs(x = "Round", y = "Chosen Arm", title = "Thompson Sampling Arm Choices Over
Time") +
  scale_color_manual(values = c("0" = "red", "1" = "green"), name = "Reward") +
  scale_y_discrete(breaks = function(x) x[seq(1, length(x), by = 5)]) + # Show every
5th arm
  theme_minimal() +
  theme(axis.text.y = element_text(angle = 45, size = 6)) # Adjust y-axis label size
and orientation

```

We started by generating batches with sizes that increase by 1262, beginning from 1262, and continuing until the maximum batch size reaches 54266. The number 1262 was chosen because it is a divisor of 54266, ensuring that we can increment the batch sizes evenly and consistently, maintaining equal-sized increments throughout the process.

#Supposedly this is the right version of batching

```{r}
# Function to split the dataset into batches based on batch size
split_into_batches <- function(dfZozo_80, batch_size) {
  n <- nrow(dfZozo_80)
  num_batches <- ceiling(n / batch_size)  # Calculate the number of batches
  batches <- list()

  for (i in 1:num_batches) {
    start_row <- ((i - 1) * batch_size) + 1
    end_row <- min(i * batch_size, n)
    batches[[i]] <- dfZozo_80[start_row:end_row, ]
  }

  return(batches)
}

# Define different batch sizes you want to test
batch_sizes <- seq(1262, 54266, by = 8834) # Example of batch sizes to test
results_list <- list()

# Loop over each batch size and run the model
for (batch_size in batch_sizes) {

  # Split the dataset into batches for the current batch size
  batches <- split_into_batches(dfZozo_80, batch_size)

  # Empty list to store the history for each batch
  history_batches <- list()

  for (i in 1:length(batches)) {
    # Thompson Sampling policy
    TS <- ThompsonSamplingPolicy$new()

    # Create bandit for the current batch
    bandit_batch <- OfflineReplayEvaluatorBandit$new(
      formula = click ~ item_id,
      data = batches[[i]],
      randomize = FALSE
    )

    # Create the agent with the Thompson Sampling policy and batch-specific bandit
    agent_batch <- Agent$new(TS, bandit_batch)

    # Set up simulator for the batch
    simulator <- Simulator$new(
      agent_batch,
      horizon = nrow(batches[[i]]),  # Use the batch size for each simulation
      do_parallel = TRUE,
      simulations = n_sim
```

```
    )

    # Run the simulation for the current batch
    history_batch <- simulator$run()

    # Get the results and add the batch number
    history_batch_data <- history_batch$data
    history_batch_data$batch_num <- i

    # Store the result for this batch
    history_batches[[i]] <- history_batch_data

    # Print batch progress
    print(paste("Completed batch", i, "for batch size", batch_size))
  }

  # Combine all batch histories into a single dataframe for the current batch size
  df_TS_combined <- do.call(rbind, history_batches) %>%
    select(t, sim, choice, reward, agent, batch_num)

  # Store results in a list with batch size as the key
  results_list[[paste0("batch_size_", batch_size)]] <- df_TS_combined
}

# You can now access the results for each batch size by calling:
# results_list[["batch_size_1262"]]
# results_list[["batch_size_2524"]]
# etc.

# Track the maximum t (time) for each simulation for batch size 1262 as an example
df_TS_zozo_combined_max_t <- results_list[["batch_size_1262"]] %>%
  group_by(sim) %>%
  summarize(max_t = max(t))

# View the max time per simulation
df_TS_zozo_combined_max_t

```


```{r}
# Function to calculate regret
calculate_regret <- function(df_TS_combined, dfZozo_80) {

  # Calculate total reward obtained by Thompson Sampling in each batch
  total_reward_TS <- df_TS_combined %>%
    group_by(batch_num) %>%
    summarize(total_reward_TS = sum(reward)) %>%
    ungroup()

  # Calculate average reward per arm from the entire dataset
  avg_reward_per_arm <- dfZozo_80 %>%
    group_by(item_id) %>%
    summarize(avg_reward = mean(click)) %>%
    ungroup()

  # Identify the arm with the highest average reward
  best_arm_avg_reward <- max(avg_reward_per_arm$avg_reward)

  # Calculate the total reward for the best arm if it had been chosen every time
  total_reward_best_arm <- df_TS_combined %>%
    group_by(batch_num) %>%
    summarize(total_reward_best = n() * best_arm_avg_reward) %>%
    ungroup()

  # Merge both total_reward_TS and total_reward_best
  regret_df <- total_reward_TS %>%
    left_join(total_reward_best_arm, by = "batch_num") %>%
```

```
    mutate(regret = total_reward_best - total_reward_TS)

  return(regret_df)
}

# Apply the regret calculation function for each batch size and store total regret
regret_summary <- data.frame(batch_size = integer(), total_regret_TS = numeric())

for (batch_size in batch_sizes) {
  # Access the combined results for the current batch size
  df_TS_combined <- results_list[[paste0("batch_size_", batch_size)]]

  # Calculate regret
  regret_df <- calculate_regret(df_TS_combined, dfZozo_80)

  # Calculate the total regret for this batch size
  total_regret_TS <- sum(regret_df$regret)

  # Append the results to the regret summary table
  regret_summary <- rbind(regret_summary, data.frame(batch_size = batch_size,
total_regret_TS = total_regret_TS))
}

# Print the regret summary in a format similar to the TS row
print("Table: Regret for TS at different batch sizes")
print(regret_summary)
```

Parameter tuning

```{r}
# Example of different priors for tuning
prior_configs <- expand.grid(alpha_0 = c(1, 10, 50, 100 ), beta_0 = c(1, 10, 50, 100))

results <- list()

for (i in 1:nrow(prior_configs)) {
  # Extract the alpha and beta values from the grid
  alpha_0 <- prior_configs$alpha_0[i]
  beta_0  <- prior_configs$beta_0[i]

  # Print the current alpha and beta values being tested
  print(paste("Testing alpha =", alpha_0, "and beta =", beta_0))

  # Set up the initial values for alpha and beta per arm
  df_alpha_beta <- data.frame(arm = arm, alpha = rep(alpha_0, length(arm)), beta =
rep(beta_0, length(arm)))

  # Run the Thompson Sampling simulation
  df_sampled <- mapply(rbeta, n_sample, df_alpha_beta$alpha, df_alpha_beta$beta)
  avg_from_sampled <- apply(df_sampled, 2, mean)

  # Find the chosen arm
  chosen_arm <- which.max(avg_from_sampled)

  # Print the chosen arm for this configuration
  print(paste("Chosen arm for alpha =", alpha_0, "and beta =", beta_0, "is arm",
chosen_arm))

  # Store the result for evaluation
  results[[i]] <- chosen_arm
}
```

```

```{r}
# Load necessary libraries
library(contextual)
library(dplyr)
library(ggplot2)

# Set seed for reproducibility
set.seed(0)

# Define number of arms and simulation parameters
n_arms <- 80
size_sim <- 54266  # Number of time steps per simulation
n_sim <- 10       # Number of simulations

# Load or define the dataset 'dfZozo_80' for OfflineReplayEvaluatorBandit
# Example dataset with rewards and arms should be provided for this bandit simulation.
# Ensure that dfZozo_80 is a dataframe with columns such as 'click' (reward) and
'item_id' (arm number)

# Define prior configurations (alpha and beta combinations)
prior_configs <- expand.grid(alpha_0 = c(50, 100, 200), beta_0 = c(50, 100, 200))

# List to store results for each prior configuration
results <- list()

# Loop over each prior configuration
for (i in 1:nrow(prior_configs)) {
  alpha_0 <- prior_configs$alpha_0[i]
  beta_0  <- prior_configs$beta_0[i]

  print(paste("Testing alpha =", alpha_0, "and beta =", beta_0))

  # Define Thompson Sampling policy with the current alpha and beta priors
  TS <- ThompsonSamplingPolicy$new(alpha = alpha_0, beta = beta_0)

  # Create a bandit using OfflineReplayEvaluatorBandit with the dataset
  bandit <- OfflineReplayEvaluatorBandit$new(formula = click ~ item_id,
                                             data = dfZozo_80,
                                             randomize = FALSE)

  # Define an agent with the current Thompson Sampling policy and the bandit
  agent <- Agent$new(policy = TS, bandit = bandit)

  # Simulate the agent for the given horizon and number of simulations
  simulator <- Simulator$new(agents = agent, horizon = size_sim, simulations = n_sim)
  history <- simulator$run()

  # Collect simulation data
  df <- history$get_data_table()

  # Store the result for this prior configuration
  results[[i]] <- df
}

# Combine results into one data frame for all prior configurations
df_results <- bind_rows(results, .id = "prior_config")

# Calculate expected mean reward for each time period (t) and prior configuration
df_expected <- df_results %>%
  group_by(prior_config, t) %>%
  summarize(expected_reward = mean(reward), .groups = 'drop')

# Plot expected mean reward over time for different prior configurations
ggplot(df_expected, aes(x = t, y = expected_reward, color = as.factor(prior_config))) +
  geom_line() +
```
```

```
  labs(title = "Expected Mean Reward over Time", x = "Time Period", y = "Expected Mean
Reward",
       color = "Prior Config") +
  theme_minimal()

```

```{r}
# Example: Tuning n_sample
n_sample_values <- c(50, 100, 200)  # Different values of n_sample to test
performance_results <- list()

for (n_sample in n_sample_values) {
  # Print the current n_sample being used
  print(paste("Using n_sample =", n_sample))

  set.seed(0)

  # Sample from the beta distributions
  df_sampled <- mapply(rbeta, n_sample, df_alpha_beta$alpha, df_alpha_beta$beta)
  avg_from_sampled <- apply(df_sampled, 2, mean)

  # Find the chosen arm
  chosen_arm <- which.max(avg_from_sampled)

  # Print the chosen arm for this n_sample
  print(paste("Chosen arm for n_sample =", n_sample, "is arm", chosen_arm))

  # Store results
  performance_results[[as.character(n_sample)]] <- chosen_arm
}

```

```{r}

# Define parameter values for tuning (number of arms is fixed at 80)
n_sample_list <- c(100, 200, 300, 400)   # Number of samples (tunable)
size_sim_list <- c(2000, 5000, 10000, 54266)  # Size of each simulation (tunable)
n_sim <- 10  # Number of simulations
max_time_steps <- 650  # Max time steps

# Loop over sample and simulation size parameters and generate graphs for each
for (n_sample in n_sample_list) {
    for (size_sim in size_sim_list) {

        # Create a new bandit for the data based on the current parameters
        bandit_Zozo_80 <- OfflineReplayEvaluatorBandit$new(formula = click ~ item_id,
data = dfZozo_80, randomize = TRUE)

        # Define the Thompson Sampling policy
        TS <- ThompsonSamplingPolicy$new()
```

```
        # Create an agent with the Thompson Sampling policy and bandit
        agent_TS_zozo_80 <- Agent$new(TS, bandit_Zozo_80)

        # Simulator to run multiple simulations
        simulator <- Simulator$new(agent_TS_zozo_80, horizon = size_sim, do_parallel =
TRUE, simulations = n_sim)

        # Run the simulation
        history_TS_zozo_80 <- simulator$run()

        # Collect results and add to the final dataframe
        df_TS_zozo_80 <- history_TS_zozo_80$data %>%
            select(t, sim, choice, reward, agent) %>%
            group_by(sim) %>%
            mutate(cumulative_reward = cumsum(reward))

        # Plot the results for each simulation
        for (sim in unique(df_TS_zozo_80$sim)) {
            df_sim <- df_TS_zozo_80 %>%
              filter(sim == sim)

            # Create the plot
            plot <- ggplot(df_sim, aes(x=t, y=cumulative_reward)) +
              geom_line(color="blue") +
              labs(x="Time Steps", y="Cumulative Reward",
                   title=paste("Cumulative Rewards for Simulation", sim,
                               "- Sample Size:", n_sample,
                               "- Simulation Size:", size_sim)) +
              theme_minimal()

            # Save the plot to a file
            ggsave(filename = paste0("cumulative_rewards_sim", sim, "_sample",
n_sample, "_size", size_sim, ".png"),
                   plot = plot, width = 7, height = 7)
        }
    }
}


```
```
```