Tutorial 2.2 - Thompson Sampling

Tayyip Altan

October 2025

In this tutorial, we will be covering the Thompson Sampling (TS) method to solve multi-armed bandit problems.

Dataset

For this tutorial, we will work with a dataset from ZOZO, the largest Japanese fashion e-commerce company. The company uses multi-armed bandit algorithms to recommend fashion items to users on its large-scale e-commerce platform.

We will work with a version of the dataset that contains 10 fashion items. Each row contains a fashion item randomly shown to a user, and if the user clicked on said fashion item. The original dataset contains +170k observations. However, in this tutorial we will be using the version with 10k observations. The fashion items were randomly shown to users, which makes this dataset suitable for evaluating our TS algorithm.

```r
# this version contains 10 arms (ZOZO)
dfZozo_10 <- read.csv('zozo_noContext_10items.csv') %>%
  select(item_id, click)
```

Below the notation and code for Thompson Sampling is displayed.

Thompson Sampling: notation

When working with the ϵ-greedy algorithm, we estimate the expected reward using the average reward up until that point. With Thompson Sampling, we take a different approach; instead of focusing solely on the expected reward, we try to find out the distribution of the reward. This is done in three steps:

1. 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.
2. At each time t, the algorithm samples n_{sample} observations from each of the distributions. The arm which has the highest average reward according to the sample is the chosen arm.
3. We then observe the reward r_t, and update the parameters of the distribution accordingly.
   - $\alpha_t = \alpha_{t-1} + r_t$
   - $\beta_t = \beta_{t-1} + 1 - r_t$

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 $n_{\text{sample}} = 100$.

```r
# set the seed
set.seed(0)

# arm index
arm <- 1:10

# alpha per arm
alpha <- c(10,2,9,3,1,8,7,4,6,5)

# beta per arm
beta <- c(1,8,3, 2,4,7,6,5,9,10)

# 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

#####
# policy_thompson: picks an arm, based on the Thompson algorithm for multi-armed bandits
#
# Arguments:
# df_alpha_beta: data.frame with three columns: arm, alpha, and beta
# n_sample: integer, the total number of samples to draw
#
#Output:
#chosen_arm;integer,indexofthearmchosen
####

policy_thompson <- function(df_alpha_beta, n_sample){

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

   # TODO: select an arm based on step 2 of TS (the arm with the highest average reward)
  # First, determine the average reward per arm
  avg_from_sampled <- apply(df_sampled, 2, mean)

  # TODO: Select the arm with the highest average reward
  chosen_arm <-

  # return the chosen arm
  return(chosen_arm)
}

# get the chosen arm from the function
chosen_arm_practice <- policy_thompson(df_alpha_beta, n_sample)
print(paste0('The chosen arm is: ', chosen_arm_practice))
```

Thompson Sampling: code

We will be simulating the performance of the Thompson Sampling algorithm on the Zozo data. For each simulation, one needs to define a (1) bandit, which draws the rewards per arm, (2) an policy/algorithm for pulling arms of the bandit, and (3) an simulator which simulates the performance of the policy/algorithm given the bandit. The bandit is the same as in Tutorial 2.1. The Thompson policy is given in Task 1, see above.

Task 2: build a simulator function. Initialize the alpha and beta parameters, and update these based on the sampled rewards.

```r
###
# sim_thompson: simulates performance of a Thompson policy
#
#   Arguments:
#
#     df: n x 3 data.frame, with column names "arm", "reward", "index"
#
#
#     n_sim: integer, number of observations used to simulate the
#            performance of the Thompson algorithm
#
#     n_sample: integer, the total number of samples to draw
#
#     interval: the number of steps after which our arm is updated.
#               For example, interval is 5 means that when an arm
#               is chosen by our approach, it is deployed for 5 steps.
#
#
#   Output: list with following
#     df_results_of_policy: n_sim x 2 data.frame, with "arm", "reward".
#                           Random sampled rewards for chosen arms
#
#     df_alpha_beta: data.frame with alphas and betas for each arm.
#
#
#
###
sim_thompson <- function(df, n_sim, n_sample, interval=1){

  # define the number of observations of all data available
  n_obs <- nrow(df)

  # define the number of arms
  n_arms <- max(df[,1])

  # Give user a warning: the size of the intended experiment is bigger than the data provided
  if(n_sim > (n_obs)){
    stop("The indicated size of the experiment is bigger than the data provided - shrink the size ")
  }

  # arm index
  arm <- 1:n_arms
```

```r
# alpha per arm
alpha <- rep(1, n_arms)

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

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

# create dataframe with data that we can sample from during policy
df_during_policy <- df

# Empty dataframe where the results of the policy are stored
df_results_policy <- data.frame(matrix(NA, nrow = n_sim, ncol = 2))
colnames(df_results_policy) <- c('arm', 'reward')

## part 2: apply Thompson algorithm, updating at interval
for (i in 1:n_sim){

  # update at interval
  if((i==1) || ((i %% interval)==0)){

     # TODO: choose arm at interval with our Thompson policy function
    chosen_arm <-

  }else{

    # TODO: In the case that we do not update, take the current arm
    chosen_arm <-
  }

  # select from the data for experiment the arm chosen
  df_during_policy_arm <- df_during_policy %>%
    filter(item_id==chosen_arm)

  # randomly sample from this arm and observe the reward
  sampled_arm <- sample(1:nrow(df_during_policy_arm), 1)
  reward <- df_during_policy_arm$click[sampled_arm]

  # important: remove the reward from the dataset to prevent repeated sampling
  index_result <- df_during_policy_arm$index[sampled_arm]
  df_during_policy_arm <- df_during_policy_arm %>% filter(index != index_result)

  # warn the user to increase dataset or downside the size of experiment,
  # in the case that have sampled all observations from an arm
  if(length(reward) == 0){
    print("You have run out of observations from a chosen arm")
    break
  }

  # get a vector of results from chosen arm (arm, reward) and add to dataframe to save the result
  df_results_policy[i,] <- c(chosen_arm, reward)
```

```
    # TODO: update alpha and beta parameters
    df_alpha_beta_at_t$alpha[df_alpha_beta_at_t$arm==chosen_arm] <-
    df_alpha_beta_at_t$beta[df_alpha_beta_at_t$arm==chosen_arm] <-

  }


  # save results in list
  results <- list(df_results_of_policy = df_results_policy,
                  df_alpha_beta = df_alpha_beta_at_t)

  return(results)

}
```

Task 3: Run 10 simulations and calculate the cumulative reward per simulation (1-10).

```
# number of observations used to simulate
n_sim <- 2500

# The number of simulations
num_simulations <- 10

# set the seed for reproducability
set.seed(0)

# Again, we create an empty dataframe where the results of the simulator are stored
df_TS_zozo_10 <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_TS_zozo_10) <- c('arm', 'reward')

# Loop over each of the 10 simulations
for (i in 1:num_simulations){

  # TODO: run Thompson simulation using the function built in Task 2 using interval = 1
  df_TS_zozo_10_temp <-

  # Append the results of the simulation to our results
  df_TS_zozo_10 <- rbind(df_TS_zozo_10, df_TS_zozo_10_temp)
}

df_TS_zozo_10 <- df_TS_zozo_10[-1,]
df_TS_zozo_10$simulation <- rep(1:num_simulations, each=n_sim)


# TODO: Calculate the cumulative reward per simulation
df_TS_zozo_10_total_reward <- df_TS_zozo_10 %>%


df_TS_zozo_10_total_reward
```

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

- **choice**: the choice made by the algorithm

- **reward**: the reward observed by the algorithm
- **agent**: column containing the name of the agent

Task 4: using this dataframe, make a plot of the average cumulative rewards for all simulations, together with the 95% confidence interval.

```r
# Create an additional column to keep track of the timestamps
df_TS_zozo_10$t <- 1:n_sim

# Max of observations.
# You may want to adjust this depending on the number of observations per simulation and the size of th
max_obs <- 9000

# Dataframe transformation
df_history_agg <- df_TS_zozo_10 %>%
  group_by(simulation) %>% # group by simulation
  mutate(cumulative_reward = cumsum(reward))%>% # calculate, per sim, cumulative reward over time
  group_by(t) %>% # group by timestep
  summarise(avg_cumulative_reward = mean(cumulative_reward), # average cumulative reward
            se_cumulative_reward = sd(cumulative_reward, na.rm=TRUE)/sqrt(num_simulations)) %>% # SE + 
  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.
```

As stated at the start, we only have 10 arms for the ZOZO data. However, there are also versions of the data with 20 arms. This is loaded for you below.

Task 5: repeat the simulation steps in the tutorial above, but now for the data with 20 arms.

```r
# this version contains 20 arms
dfZozo_20 <- read.csv('zozo_noContext_20items_tiny.csv')%>%
  select(item_id, click)
dfZozo_20$index <- 1:nrow(dfZozo_20)
```

```r
n_sim <- 2500
num_simulations <- 10

# set the seed
set.seed(0)

# Again, we create an empty dataframe where the results of the simulator are stored
df_TS_zozo_20 <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_TS_zozo_20) <- c('arm', 'reward')

# Loop over each of the 10 simulations
for (i in 1:num_simulations){

  # TODO: run Thompson simulation using the function built in Task 2 fr the dataset with 20 arms
  df_TS_zozo_20_temp <-

  # Append the results of the simulation to our results
  df_TS_zozo_20 <- rbind(df_TS_zozo_20, df_TS_zozo_20_temp)
}

df_TS_zozo_20 <- df_TS_zozo_20[-1,]
df_TS_zozo_20$simulation <- rep(1:num_simulations, each=n_sim)
```

Task 6: Make a plot that compares the average cumulative reward (with 95% confidence interval) for 10 and 20 arms. For which version does Thompson Sampling perform better?.}

```
 #set max observations to create fair comparison across simulations
 max_obs <- 2000
 df_TS_zozo_20$t<-1:n_sim

# set the agent variable to number of
df_history_agg_20 <- df_TS_zozo_20 %>%
  group_by(simulation)%>% # group by simulation
  mutate(cumulative_reward = cumsum(reward))%>% # calculate cumulative reward
  group_by(t) %>% # group by timestep t
  summarise(avg_cumulative_reward = mean(cumulative_reward), # calculate average cumulative reward
            se_cumulative_reward = sd(cumulative_reward, na.rm=TRUE)/sqrt(num_simulations)) %>% # calcu
  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)

# Combine the dataframes
df_history_agg <- bind_rows(df_history_agg %>% mutate(no_arms='10'), df_history_agg_20 %>% mutate(no_arr

# TODO: Make the following two plots:
# 1: A plot that compares the average cumulative rewards over time for 10 and 20 arms using the df_hist
# 2: The plot as defined in (1) together with the 95\% confidence interval.
```