Tutorial 2 - Thompson Sampling and UCB

Floris Holstege, Bernhard van der Sluis

January 2024

In this problem set, we will be covering Thompson Sampling and Upper Confidence Bound (UCB) methods to solve multi-armed bandit problems.

Dataset

Today, we will work with two datasets. For the part about Thompson sampling, we work with a dataset from ZOZO. ZOZO is the largest Japanese fashion e-commerce company. The company uses multi-armed bandit algorithms to recommend fashion items to users in a large-scale fashion e-commerce platform. For the part about UCB, we work with the Yahoo dataset from the first tutorial.

We will first work with a version of the dataset which contains 10 fashion items. Each row contains an fashion item randomly shown to a user, and if the user clicked on said fashion item. It contains +300k observations. The fashion items were randomly shown to users, which makes this dataset suitable for evaluating our TS algorithm.

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

# reads in full csv of Yahoo dataset
dfYahoo <- read.csv('yahoo_day1_10arms.csv')[,-c(1,2)]

# selects the two relevant columns from Yahoo dataset; arm shown to user and reward observed
dfYahoo_for_sim <- dfYahoo %>% select(arm, reward)
dfYahoo_for_sim$index <- 1:nrow(dfYahoo_for_sim)
```

Below we will first introduce the notation and code for Thompson Sampling. The notation and code for UCB follows as second.

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 expected reward, we try to find out the distribution of the reward. This is done in three steps:

1. Let $Q_t(a)$ follow a beta distribution: $f(x : \alpha_t, \beta_t) = \frac{\Gamma(\alpha_t+\beta_t)}{\Gamma(\alpha_t)\cdot\Gamma(\beta_t)} x^{\alpha_t-1}(1-x)^{\beta_t-1}$. At $t = 0$, every arm follows the same beta distribution with $\alpha_0 = 1, \beta_0 = 1$.
2. At each 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 $n_{\text{sample}} = 100$.

```
# 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

# TODO: select arm based on thompson sampling
```

Thompson Sampling: code

We will be simulating the performance of the Thompson Sampling algorithm on the Zozo data.

To do so easily, we are using the *contextual* package in R. The documentation for this package can be found here. This package allows for easy simulation and implementation of various multi-armed bandit algorithms. 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.

Below, we go over an example of how one can use the package. Let's start by simply applying the Thompson Sampling algorithm to the ZOZO data. We (1) define a bandit, (2) a policy/algorithm and (3) and agent.

```
# 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 20 arms,
#   formula = click ~ item_id,
```

```
#   no randomization
bandit_Zozo_10 <- OfflineReplayEvaluatorBandit$new(formula = click ~ item_id,
                                                   data = dfZozo_10,
                                                   randomize = FALSE)

?OfflineReplayEvaluatorBandit
# lets generate a 10 simulations, each of size 100.000,
size_sim=100000
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_10 <-  Agent$new(TS, # add policy
                               bandit_Zozo_10) # 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_10, # set our agent
                                    horizon= size_sim, # set the sizeof 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_10            <- simulator$run()

# gather results
df_TS_zozo_10 <- history_TS_zozo_10$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.

```
df_TS_zozo_10_max_t <- df_TS_zozo_10%>%
  group_by(sim) %>% # group by per agent
  summarize(max_t = max(t)) # get max t

df_TS_zozo_10_max_t
```

```
## # A tibble: 10 x 2
##      sim max_t
##    <int> <int>
##  1     1  9874
##  2     2 10196
##  3     3  9946
##  4     4 10275
##  5     5 10053
##  6     6  9968
##  7     7 10125
##  8     8 10068
##  9     9  9901
## 10    10 10076
```

Task 2: answer below: why is the maximum number of observations different for each simulation?

Answer:

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

- t: the step at which a choice was made
- **sim**: the simulation for which we observe results
- **choice**: the choice made by the algorithm
- **reward**: the reward observed by the algorithm
- **agent**: column containing the name of the agent

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

```
# Max of observations. Depends on the number of observations per simulation
max_obs=9000


# TODO: add plot of the average cumulative rewards for all simulations,
# 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 and 40 arms. These are loaded for you below.

Task 4: repeat the steps in the tutorial above, but now for the data 40 arms. Make a plot that compares the average cumulative reward (with 95% confidence interval) for 10 and 40 arms. For which version does Thompson Sampling perform better?.

```
# this version contains 20, 40 arms
dfZozo_40 <- read.csv('zozo_noContext_20items.csv')%>%
  select(item_id, click)
```

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

# TODO: add plot that compares average cumulative reward
# (with 95\% confidence interval) for 10 and 40 arms
```

UCB: notation

As you saw in the lecture, UCB methods decide on the arm to pick using an 'optimistic' estimate of the reward - e.g. which arm has the most potential to yield a high reward? This is done by picking the arm that has the highest combination of (1) the estimated reward, and (2) an bonus that rewards the arm for being underexplored, denoted as $U_t(a)$. For the UCB algorithm, this becomes

$$a_t = \text{argmax}_{a \in \mathcal{A}} Q_t(a) + c \cdot U_t(a). \tag{1}$$
$$\tag{2}$$

In the general case of the UCB algorithm, $U_t(a) = \sqrt{\frac{log(t)}{N_t(a)}}$, where $N_t(a)$ is the number of times arm a has been pulled so far. $N_t(a)$ grows, the exploration reward becomes smaller. The intuition is here is that as we gather more info about the arm by pulling it, we become more certain about its average reward. The parameter c determines the degree of exploitation vs. exploration - if c is high, we are more likely to explore an arm with potential.

Task 5: select the arm according to the UCB algorithm based on the data below . Here, the code already provides a dataframe with per arm (1) the average reward, (2) the number of pulls thus far, and (3) an given value for t. Use this dataframe to determine which arm should be pulled if $c = 0.1$. The selected arm should be 8.

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

# get per item, the average reward and the number of items observed
dfYahoo_summary <- dfYahoo %>%
  group_by(arm) %>%
  summarise(avg_reward = mean(reward),
            n_pulls= n())

# the t in this case is simply the total of observations
t <- sum(dfYahoo_summary$n_pulls)
c= 0.1

# TODO: select arm based on UCB criterion
```

UCB: code

We will be simulating the performance of the UCB algorithm on the Yahoo! data, again with the contextual package.

```
# 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 20 arms,
#   formula = reward ~ arm,
#   no randomization
bandit_yahoo <- OfflineReplayEvaluatorBandit$new(formula = reward ~ arm,
                                                 data = dfYahoo,
                                                 randomize = FALSE,
                                                 )


# lets generate a 10 simulations, each of size 100.000,
size_sim=100000
n_sim=10

# here we define the UCB policy object
# for some reason, the contextual package uses the 'UCB2Policy' tag for standard UCB algorithm
# in the package, 'alpha' is what we refer to as c
UCB_01          <- UCB2Policy$new(alpha=0.1)

# the contextual package works with 'agent' objects - which consist of a policy and a bandit
UCB_01_agent <- Agent$new(UCB_01, # add our UCB1 policy
                          bandit_yahoo) # add our 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(UCB_01_agent, # set our agent
                                    horizon= size_sim, # set the sizeof each simulation
                                    do_parallel = TRUE, # run in parallel for speed
                                    simulations = n_sim, # simulate it n_sim times
                                  )


# run the simulator object
history_UCB_01            <- simulator$run()

# gather results
df_Yahoo_UCB_01 <- history_UCB_01$data %>%
  select(t, sim, choice, reward, agent)
```

In the dataframe *df_Yahoo_UCB_01* we have gathered the results of the UCB policy for $c = 0.1$.

Task 6: repeat the steps of this tutorial, but now for $c = 0.5$. Make a plot that compares the UCB policy for $c = 0.1$, $c = 0.5$. Compare the policies based on average cumulative reward. Can you conclude which one performs better, and if so why?. Set the *horizon* argument to 10.000

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

# TODO: make plot to compare UCB policy for c=0.1, c=0.5
```

Task 7: compare the UCB policy for $c = 0.1$ to an ϵ-greedy policy where $\epsilon = 0.1$. Which one does better? Why do you think this might be?. Repeat this exercise again with *horizon* of 10.000. Hint: use the *EpsilonGreedyPolicy$new(epsilon = 0.1)* function from the *contextual* package.

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

# TODO: compare UCB policyfor $c=0.1$   to epsilon-greedy policy where $\epsilon=0.1$
```

Task 8: to better understand the ϵ-greedy policy and UCB algorithm, plot for each the cumulative % of the time a certain arm is chosen. Which arm is chosen most often? does this differ between the two approaches?

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

# up until this observation
max_obs = 2500


# TODO: Plot the average cumulative % of arms chosen per UCB, epsilon greedy
```