Tutorial 2.3 - Upper Confidence Bound

Tayyip Altan

October 2025

In this tutorial, we will be covering the Upper Confidence Bound (UCB) method to solve multi-armed bandit problems.

Dataset

For this tutorial, we will work with the Yahoo dataset, which was also used in tutorial 2.1.

```
# reads in full csv of Yahoo dataset
dfYahoo <- read.csv('yahoo_day1_10arms_tiny.csv')[,-c(1,2)]

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

UCB: notation

As covered 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, $Q_t(a)$, and (2) a bonus that rewards the arm for being underexplored, denoted as $U_t(a)$. The UCB algorithm picks the arm that maximizes the sum of these two factors:

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

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.

Task 1: 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)
```

```r
c <- 0.1

#####
# policy_ucb : picks an arm, based on the UCB algorithm for multi - armed bandits
#
# Arguments :
# df: data.frame with two columns: arm, reward
# c : float, UCB penalty parameter
##
# Output :
# chosen_arm ; integer, index of the arm chosen
####
policy_ucb <- function(df, c){
  # get per item, the average reward and the number of items observed
  dfsummary <- df %>%
    group_by(arm) %>%
    summarise(avg_reward = mean(reward),
              n_pulls= n())

  # the t in this case is simply the total of observations
  t <- sum(dfsummary$n_pulls)

  # TODO: Select an arm based on the UCB criterion (using equation (1))
  ucb_reward <- dfsummary$avg_reward + c*sqrt(log(t)/dfsummary$n_pulls)
  chosen_arm <- which.max(ucb_reward)
  return(chosen_arm)
  }

# get the chosen arm from the function
ucb_ichosen_item <- policy_ucb(dfYahoo, c)
print(paste0('The chosen arm is: ', ucb_ichosen_item))
```

UCB: code

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

```r
###
# sim_ucb: simulates performance of a UCB policy
#
#   Arguments:
#
#     df: n x 3 data.frame, with column names "arm", "reward", "index"
#
#
#     n_before_sim: integer, number of observations (randomly sampled)
#                   before starting the UCB algorithm
#
#
#     n_sim: integer, number of observations used to simulate the
#            performance of the epsilon-greedy algorithm
#
```

```r
#     c: float, UCB penalty parameter
#
#     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_sample_of_policy: data.frame with sample used to evaluate policy.
#
#
#
###
sim_ucb <- function(df, n_before_sim, n_sim, c, interval=1){

  ## Part 1: create two dataframes, one with data before start of policy, and one with data after

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

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

  # Next we prepare our dataframes using a function from helper_functions.R
  prepped_data <- prepare_dataframes(df, n_obs, n_before_sim, n_sim)

  # Access individual data frames
  df_results_policy <- prepped_data$df_results_policy
  df_during_policy  <- prepped_data$df_during_policy
  df_results_at_t   <- prepped_data$df_results_at_t

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

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

      # TODO: pick an arm according to the UCB policy using the function from Task 1
      chosen_arm <- policy_ucb(df_results_at_t,c)
      current_arm <- chosen_arm

    }else{
      # TODO: if not updating, take current arm
      chosen_arm <- current_arm
    }

    # select from the data for experiment the arm chosen
    df_during_policy_arm <- df_during_policy %>%
```

```r
      filter(arm==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$reward[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)
    result_policy_i <- c(chosen_arm, reward)

    # add to dataframe to save the result
    df_results_policy[i,] <- result_policy_i

    # TODO: combine to dataframe with all results
    df_results_at_t <- rbind(df_results_at_t, result_policy_i)

  }


  # save results in list
  results <- list(df_results_of_policy = df_results_policy,
                  df_sample_of_policy = df[-index_before_sim,])

  return(results)
}
```

Task 2: Run 10 simulations and calculate the cumulative reward per simulation (1-10).

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

# The number of simulations
num_simulations <- 10
c <- 0.1

# set the seed
set.seed(0)

# Create an empty dataframe where the results of storing the simulator are stored
df_Yahoo_UCB_01 <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_Yahoo_UCB_01) <- c('arm', 'reward')

# Loop to get 10 simulations
```

```
for (i in 1:num_simulations){

  # TODO: run UCB simulation using the function built in the previous task
  df_Yahoo_UCB_01_temp <- sim_ucb(dfYahoo, n_before_sim=100, n_sim=n_sim, c=c, interval=1)[[1]]

  # Append the results to our dataframe
  df_Yahoo_UCB_01 <- rbind(df_Yahoo_UCB_01, df_Yahoo_UCB_01_temp)
}

df_Yahoo_UCB_01 <- df_Yahoo_UCB_01[-1,]
df_Yahoo_UCB_01$simulation <- rep(1:num_simulations, each=n_sim)
```

Task 3: repeat the steps of this tutorial, but now for $c = 0.5$.

```
c <- 0.5

# set the seed
set.seed(0)

# Empty dataframe where the results of storing the simulator are stored
df_Yahoo_UCB_05 <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_Yahoo_UCB_05) <- c('arm', 'reward')

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

  # TODO: Run the UCB algorithm for c = 0.5
  df_Yahoo_UCB_05_temp <- sim_ucb(dfYahoo, n_before_sim=100, n_sim=n_sim, c=c, interval=1)[[1]]

  # Append the results to our dataframe to keep track of the results
  df_Yahoo_UCB_05 <- rbind(df_Yahoo_UCB_05, df_Yahoo_UCB_05_temp)
}
df_Yahoo_UCB_05 <- df_Yahoo_UCB_05[-1,]
df_Yahoo_UCB_05$simulation <- rep(1:num_simulations, each=n_sim)
```

Task 3: 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?

Your answer to Task 3 here:

```
# set max observations to create fair comparison across simulations
max_obs <- 2500

df_Yahoo_UCB_01$t <- 1:n_sim
df_history_agg_01 <- df_Yahoo_UCB_01 %>%
  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)

df_Yahoo_UCB_05$t <- 1:n_sim
df_history_agg_05 <- df_Yahoo_UCB_05 %>%
  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_ucb <- bind_rows(df_history_agg_01 %>% mutate(c='0.1'), df_history_agg_05 %>% mutate(c='(

# TODO: make a plot using ggplot to compare UCB policy for c=0.1 and c=0.5
# 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.

ggplot(data=df_history_agg_ucb, aes(x=t, y=avg_cumulative_reward, color =c))+
  geom_line(size=1.5)+ # create line
  geom_ribbon(aes(ymin=ifelse(cumulative_reward_lower_CI<0, 0,cumulative_reward_lower_CI) , # create co
                  ymax=cumulative_reward_upper_CI,
                  fill = c,

  ),
  alpha=0.1)+
  labs(x = 'Time', y='Cumulative Reward', color ='c', fill='c')+
  theme_bw()+
  theme(text = element_text(size=16))
```

Task 4: Suppose we compare the UCB policy for $c = 0.1$ to an ϵ-greedy policy where $\epsilon = 0.1$, and find that the ϵ-greedy policy works better in this case. Why do you think this might happen?.

Answer: Using the ϵ-greedy policy where $\epsilon = 0.1$ scores on average better. This might be because exploring random arms is better than getting 'stuck' in arms that might seem to have potential.