Tutorial 3 - Contextual bandits

Bernhard van der Sluis

January 2025

All previous algorithms that we considered were only based on the observed past rewards, and the weighing of uncertainty around those. But obviously, there is a myriad of other forms of information that we can use to assess which arm will yield the most reward. A contextual multi-armed bandit uses this information (referred to as context) to decide which arm to pick at each timestep. This tutorial is about a contextual version of the UCB algorithm called linear UCB, and it is illustrated with an example of cryptocurrency markets.

Theory: contextual UCB

All notation is the same as in the previous tutorial. Recall the UCB method for selecting an arm a_t:

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

where $U_t(a) = \sqrt{\frac{log(t)}{N_t(a)}}$, and $Q_t(a) = \frac{R_1+R_2+...+R_{N_t(a)}}{N_t(a)}$. In this case, our estimated $Q_t(a)$ is just based on the sample average.

Now that we want to use context to determine our $Q_t(a)$. With the linear UCB algorithm, we aim to model a linear relationship between contextual features and the reward per arm. Let $\mathbf{X}_{a,t}$ be an $N_t(a)$ x m matrix, with m contextual features for arm a, observed up until moment t. Let $\mathbf{R}_{a,t}$ be a column vector containing the $N_t(a)$ previous rewards for action a, and $\mathbf{x}_{a,t}$ a specific row vector of user features at t for action a. Linear UCB estimates $Q_t(a)$ as follows:

$$a_t = \mathbf{x}'_{a,t}\hat{\boldsymbol{\theta}}_a + c\sqrt{\mathbf{x}'_{a,t}\mathbf{A}_a^{-1}\mathbf{x}_{a,t}}, \tag{2}$$

$$\hat{\theta}_a = \mathbf{A}_a^{-1}\hat{\beta}_a, \tag{3}$$

$$\mathbf{A}_{a_t} = \mathbf{A}_{a_t-1} + \mathbf{X}'_{a,t}\mathbf{X}_{a,t}, \tag{4}$$

$$\hat{\boldsymbol{\beta}}_{a,t} = \hat{\boldsymbol{\beta}}_{a,t-1} + \mathbf{X}'_{a,t}\mathbf{R}_{a,t}. \tag{5}$$

Let's unpack several parts of this equation

$$a_t = \arg\max_a [\overbrace{\mathbf{x}'_{a,t}\hat{\boldsymbol{\theta}}_a}^{\text{Est. reward}} + c\overbrace{\sqrt{\mathbf{x}'_{a,t}\mathbf{A}_a^{-1}\mathbf{x}_{a,t}}}^{\text{Uncertainty}}] \tag{6}$$

- The estimated reward for an arm a is based upon a linear regression between the contextual features ($\mathbf{X}_{a,t-1}$) and the rewards ($\mathbf{R}_{a,t-1}$) up until that point. Using the estimated coefficients of this regression ($\hat{\boldsymbol{\beta}}_a$), we estimate our expected reward for a new observation at t.

- The uncertainty is measured with the standard deviation of the reward for an arm, $\sqrt{\mathbf{x}'_{a,t}\mathbf{A}^{-1}_{a,t-1}\mathbf{x}'_{a,t}}$.

Note that by including an intercept in $\mathbf{X}_{a,t-1}$, one still can incorporate the information from the average reward of pulling an arm. Overall, the linear UCB algorithm is simply an extension of the UCB algorithm that, in addition to the average reward, incorporates information from contextual features.

Example of usefulness of context: trading cryptocurrencies

To illustrate the usefulness of adding context, let's compare the UCB algorithm with the linear UCB algorithm for the purpose of trading cryptocurrencies. You are given data in the 'df_cryptocurrencies.csv' file. This file contains the following information:

- **Symbol**: reflects for which cryptocurrency the information is recorded
- **Returns**: the returns for holding that cryptocurrency at the specific day
- **Date**: day and time for which the returns are recorded
- **OBV**: On-Balance Volume (OBV), is a measure for changes in the the amount of purchases (volume) for a cryptocurrency. Let P_t denote the price at t and Volume$_t$ denote the volume at t. OBV then is calculated as follows:

$$\text{OBV}_t = \begin{cases} \text{if } P_{t-1} < P_t & OBV_{t-1} + \text{Volume}_t \\ \text{if } P_{t-1} \geq P_t & OBV_{t-1} - \text{Volume}_t \end{cases} \tag{7}$$

- **roll returns week**: average returns over the last 5 trading days (a week)
- **prev returns**: returns of the coin for the previous trading day

```r
# Packages required for subsequent analysis. P_load ensures these will be installed and loaded.
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```r
pacman::p_load(knitr,
               ggplot2,
               png,
               tidyverse,
               rlist,
               lubridate,
               zoo,
               roll)

# get the dataframe with coins
df_coins <- read.csv('df_cryptocurrencies.csv')
df_coins$X <- NULL
df_coins$obv <- df_coins$obv / 1000000
```

Task 1: implement the Linear UCB policy. COmplete the function below according to the formulae above.

```r
policy_linearucb <- function(context, vars, theta, c, i){

  expected_rewards <- rep(0, context$n_arms)

  Xt <- context$X[context$X$Date==i,vars]
  for (arm in 1:context$n_arms) {

        Xa          <- as.numeric(Xt[Xt$Symbol == arm,vars[-1]])
        A           <- theta$A[[arm]]
        b           <- theta$b[[arm]]

        A_inv       <- solve(A)

        theta_hat   <- A_inv %*% b

        # TODO: compute expected reward for linear UCB
  }
  action   <- which(expected_rewards==max(expected_rewards))
  if(length(action)>1){
    action <- sample(action, 1)
  }
  return(action)
  }
```

Task 2: build a simulator function for linear UCB.

```r
###
# sim_linearucb: simulates performance of a UCB policy
#
#   Arguments:
#
#     df: n x 8 data.frame
#
#
#     n_sim: integer, number of observations used to simulate the
#            performance of the epsilon-greedy algorithm
#
#     c: float, linear 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_linearucb <- function(df, vars, 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)){
    stop("The indicated size of the experiment is bigger than the data provided - shrink the size ")
  }

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

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

  # initialize linear UCB parameters
  d <- length(vars)-1
  arms <- unique(df$Symbol)
  n_arms <- length(arms)
  context <- list('X'= df, 'd'=d, 'arms'=arms, 'n_arms'=n_arms)
  theta <- list('A' = replicate(n_arms, diag(1,d,d), simplify = F), 'b' = replicate(n_arms, rep(0,d), s

  ## part 2: apply UCB algorithm, updating at interval
  i = 1
  while(i <= n_sim){

    # TODO: choose arm at interval with linear UCB policy

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

    # observe the reward
    sampled_arm <- df_during_policy_arm$index
    reward <- df_during_policy_arm$returns

    # 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: update regression parameters for chosen_arm

    # onto the next
    i <- i + 1

  }


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

  return(results)
}
```

The code below runs 10 simulations for the linear UCB simulator.

```
# gather results
n_sim <- 2000
n_agents <- 10
c <- 0.1
vars <- c('Symbol','obv', 'roll_returns_week', 'prev_returns')

# set the seed
set.seed(0)
chunk_indices <- seq(1, nrow(df_coins), by = 4)

# dataframe where the results of storing the simulator are stored
df_linUCB <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_linUCB) <- c('arm', 'reward')
for (i in 1:n_agents){
  # with interval=1
  shuffled_chunks <-sample(chunk_indices)
  df_coins_temp <- df_coins[as.vector(sapply(shuffled_chunks, function(x) x: (x+4 -1))),]
  df_coins_temp$Date <- rep(1:(length(df_coins_temp$Date)/4), each =4)
  df_linUCB_temp <- sim_linearucb(df_coins_temp, vars, n_sim=n_sim, c=c, interval=1)[[1]]

  df_linUCB <- rbind(df_linUCB, df_linUCB_temp)
}
df_linUCB <- df_linUCB[-1,]
df_linUCB$agent <- rep(1:n_agents, each=n_sim)
```

The code below plots the results for the linear UCB simulations.

```
df_linUCB$t <- 1:n_sim

# Max of observations. Depends on the number of observations per simulation
max_obs<-n_sim
```

```r
df_history_agg <- df_linUCB %>%
  group_by(agent)%>% # 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(n_agents)) %>% # SE + Confide
  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)


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

# create ggplot object
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='Cumulative Reward', color='Metric')+ # add titles
  scale_color_manual(values=legend)+ # add legend
  theme_bw()+ # set the theme
  theme(text = element_text(size=16)) # enlarge text
```

Task 3: compare the performance of linear UCB to UCB (without context).The UCB algorithm is already implemented for you. Make a plot of the cumulative reward (CR) over time. The cumulative reward should be calculated as:

$$\text{CR}_T = \sum_{t=1}^{T} log(1 + r_t) \tag{8}$$

Also plot the 95%confidence interval. For how many observations should you make the cumulative reward? and which of the two algorithms performs best?

```r
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)

  ucb_reward <- dfsummary$avg_reward + c*sqrt(log(t)/dfsummary$n_pulls)
  chosen_arm <- which.max(ucb_reward)
  return(chosen_arm)
}
```

```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
#
#     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 ")
  }

  # find n_before_sim random observations to be used before start policy
  index_before_sim <- sample(1:n_obs,n_before_sim)

  # using indexing, create dataframe with data before start policy
  df_before_policy <- df[index_before_sim,]

  # save dataframe with all the results at t - to begin with those before the policy
  df_results_at_t <- df_before_policy %>% select(arm, reward)

  # create dataframe with data that we can sample from during policy
```

```r
df_during_policy <- df[-index_before_sim,]

# dataframe where the results of storing 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 UCB algorithm, updating at interval
i = 1
while(i <= n_sim){

  # update at interval
  if((i==1) || ((i %% interval)==0)){
    # pick an arm according to the UCB policy
    chosen_arm <- policy_ucb(df_results_at_t,c)
    current_arm <- chosen_arm
  }else{
    # 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 %>%
    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

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

  # onto the next
  i <- i + 1

}
```

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

  return(results)
}
```

```r
# gather results
n_sim <- 2000
n_agents <- 10
c <- 0.1

# set the seed
set.seed(0)

df_coins_UCB <- df_coins %>% select(Symbol, returns, index)
colnames(df_coins_UCB) <- c('arm', 'reward', 'index')

# dataframe where the results of storing the simulator are stored
df_UCB <- data.frame(matrix(NA, nrow = 1, ncol = 2))
colnames(df_UCB) <- c('arm', 'reward')
for (i in 1:n_agents){
  # with interval=1
  df_UCB_temp <- sim_ucb(df_coins_UCB, n_before_sim=100, n_sim=n_sim, c=c, interval=1)[[1]]

  df_UCB <- rbind(df_UCB, df_UCB_temp)
}
df_UCB <- df_UCB[-1,]
df_UCB$agent <- rep(1:n_agents, each=n_sim)
```

```r
# TODO: add plot of the average cumulative rewards for all simulations,
# together with the 95\% confidence interval. Do this for both UCB (without context) and linear UCB.
```

(Bonus) Task 4: can you make the linear UCB algorithm even better by adding context? create a variable of your own, and assess if it makes the algorithm perform better. Explain why you added the variable.