Problem Set 1 - Epsilon-Greedy and Calculating Regret

Floris Holstege, Bernhard van der Sluis

January 2024

Introduction

In this problem set, you will be implementing two concepts.

- **The ϵ-Greedy Algorithm**: In short, this algorithm picks in ϵ % of cases a random arm (exploration). In 1-ϵ- % of cases, pick the arm with at that point the highest average reward (exploitation).
- **Regret calculation**: in this case, regret means the reward the algorithm did not obtain because it did not play the optimal arm. This means the difference between the rewards obtained by pulling the chosen arms (in this case, the arms selected by the ϵ-Greedy algorithm) and the rewards that would have been obtained had we pulled the optimal arms.

Before going further: check that you have R, R markdown and tinytex correctly installed. Tutorial 0 provides instructions for doing so.

Loading libraries

Before starting the problem set, make sure you have all the libraries installed that are needed. Simply run this chunk below.

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

```
## Warning: package 'pacman' was built under R version 4.3.2
```

```
pacman::p_load(dplyr,
               tidyr,
               ggplot2,
               reshape2,
               latex2exp
               )
```

Dataset

We will explore the concepts from this problem set with a dataset from Yahoo!. This dataset contains for 10 articles when that article was shown to a user, and if the user clicked on said article. The data is from a single day and contains +400k observations. It contains three columns:

- **arm**: the index of the article shown to the user (1-10)
- **reward**: 0 if an user did not click on the article, 1 if an user did click on the article
- **index**: unique id of the arm-reward combination.

Each observation is an arm and the respective reward for showing that arm to an user. The articles were randomly shown to users, which makes this dataset suitable for evaluating our ϵ-Greedy Algorithm.

```r
# 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)
```

ϵ-Greedy Algorithm: notation

Before delving into the algorithm, let's recap notation used in the lecture. $a \in \mathcal{A}$ indicates an arm available to our algorithm from the set of all available arms $\mathcal{A}$. We denote a_t as the chosen arm at time t. For reward r, we observe at t the reward r_t for arm a. Our goal is to choose the action with the highest reward: $Q^*(a) = E(r_t|a_t = a)$. The problem is that we do not know r_t, and thus need to estimate $Q^*(a)$. Let this estimate be called $Q_t(a)$. The most basic way of making this estimate is by simply taking the average reward for an arm a up until that point. Let $N_t(a)$ be the number of times a is chosen up until moment t. Let $R_1, R_2, ..., R_{N_t(a)}$ be the rewards for a. Then:

$$Q_t(a) = \frac{R_1 + R_2 + ... + R_{N_t(a)}}{N_t(a)} \tag{1}$$

In the case of a greedy algorithm, our chosen arm $a_t = \text{argmax}_{a \in \mathcal{A}} Q_t(a)$. In the case of an ϵ-Greedy Algorithm, we pursue the greedy option in $(1 - \epsilon)\%$ of cases, and in $\epsilon\%$ of cases we select a random arm. Thus, ϵ determines the degree of exploitation (greedy) vs. exploration (random choice). More formally, in the case of an ϵ-Greedy Algorithm, our chosen arm becomes:

$$a_t = \begin{cases} \text{argmax}_{a \in \mathcal{A}} Q_t(a) & \text{with prob. } 1 - \epsilon \\ a \in_R \mathcal{A} & \text{with prob. } \epsilon \end{cases} \tag{2}$$

where $a \in_R \mathcal{A}$ denotes a random chosen arm from the set of available arms $\mathcal{A}$.

ϵ-Greedy Algorithm: code

We will code up our own version of the ϵ-Greedy algorithm in two steps. First, you will code a function that based on a set of arms and rewards, selects an arm according to the ϵ-Greedy algorithm. Second, this tutorial will provide a function for simulating the ϵ-Greedy algorithm on the Yahoo dataset.

Task 1: finish the function specified below. Given a set of arms and rewards, we want to write a function, to be called 'policy_greedy', that selects an arm. It should do so in line with equation (2).

IMPORTANT: The outline of the function is specified below - you only have to fill in the part that states 'TODO'.

Your function should return an integer which indexes which arm (1-10) is the chosen arm. In order to check if your code works, set $\epsilon = 0$ - it then should return 3. Then, set $\epsilon = 0.5$. Now, in 50% of cases it should return 3, and in 50% cases another random number.

```r
# grab first 100 observations, create dataframe to practice for policy_greedy() function
df_practice <- dfYahoo_for_sim[1:100,]

# set value of epsilon parameter
eps = 0.
```

```
#####
# policy_greedy : picks an arm, based on the greedy algorithm for multi - armed bandits
#
# Arguments :
#  df: data.frame with two columns: arm, reward
#  eps : float, percentage of times a random arm needs to be picked
#  n_arms: integer, the total number of arms available. Standard value is 10.
#
# Output :
# chosen_arm ; integer, index of the arm chosen
####
policy_greedy <- function(df, eps, n_arms=10){

  # TODO: write function to select arm based on greedy criterion

  # return the chosen arm
  return(chosen_arm)
}
```

Now that we have this function, we can use it to simulate an ϵ-Greedy algorithm for the Yahoo dataset. Before the simulation starts, there is an initial period in which we gather some information about the arms. In this period, users are randomly exposed to articles. Based on this initial information, we then start our ϵ-Greedy algorithm to pick the articles. For $n_{\text{before simulation}}$ steps we gather information, and for $n_{simulation}$ steps we simulate our algorithm.

The function on the next page simulates the performance of our ϵ-Greedy algorithm.

Task 2: finish the function specified below.

```
###
# sim_greedy: simulates performance of an epsilon-greedy 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 epsilon-greedy algorithm
#
#     n_sim: integer, number of observations used to simulate the
#            performance of the epsilon-greedy algorithm
#
#     epsilon: float between 0 and 1. In  epsilon%  of cases,
#              randomly explore other arms. In 1-epsilon %, pick arm
#              that has highest average reward up until that point.
#
#     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
#
```

```r
#     df_sample_of_policy: data.frame with sample used to evaluate policy.
#
#
#
###
sim_greedy <- function(df, n_before_sim,n_sim,epsilon, 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
  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: appply epsilon-greedy algorithm, updating at interval
  i = 1
  while(i <= n_sim){


    # TODO: select the arm with your policy_greedy function


    # 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){
      stop("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 the data.frame df_results_at_t to incorporate the new information

    # onto the next
    i <- i + 1

  }


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

  return(results)

}
```

Task 3: simulate the epsilon greedy algorithm. With the 'sim_greedy' function, run the epsilon greedy algorithm for the following parameters: $n_{\text{before simulation}} = 500$, $n_{simulation} = 1000$, $\epsilon = 0.1$ and $\epsilon = 0.25$. Calculate the total reward based on the returned 'df_results_of_policy'.

```
# set parameters: observations before start simulation, then how many observations for simulation
n_before_sim = 500
n_sim = 1000

# set the seed
set.seed(0)

# with epsilon = 0.1,
#TODO: use sim_greedy to simulate results for epsilon = 0.1


# with epsilon = 0.25
#TODO: use sim_greedy to simulate results for epsilon = 0.25
```

Task 4: plot performance for t Compare the performance of the algorithm for $\epsilon = 0.1$ and $\epsilon = 0.25$ using a plot. The x-axis should show the steps t, and the y-axis the cumulative reward at point t. Which one of these performs better? give a general explanation why we would expect this version of the algorithm to perform better?

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

```
# TODO: create plot to show performance for epsilon = 0.1, epsilon = 0.25
```

Task 5: the effect of different parameters

Let's keep $\epsilon = 0.1$. What happens if you change $n_{\text{before simulation}}$ to 1000? Give an explanation why we observe the change in performance.

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

# TODO: Create the same plot as previously for epsilon=0.1, and n_before_sim=100, n_before_sim=1000
```

Extra: if you have time left, consider implementing a grid-search to find the best combination for $n_{\text{before simulation}}$ and ϵ.

Calculating regret

The unobserved best action-value is the average reward for arm a: $q^*(a) = E[r|a]$. The optimal value $V^* = \max_{a \in \mathcal{A}} \quad q^*(a)$. Regret is the opportunity loss given what we could have received, and what we did receive. Total regret at t is then

$$L_t = E[\sum_{\tau=1}^{t} V^* - Q(a_\tau)] \tag{3}$$

Intuitively, it measures the difference between picking the arm with the best average reward, and the arms picked by our algorithm.

Task 6: calculate regret for $\epsilon = 0.1$

Based on equation (3), calculate the regret for the greedy algorithm when $\epsilon = 0.1$.

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

# TODO: calculate regret for the greedy algorithm when epsilon = 0.1
```