Tutorial 2.1 - Epsilon-Greedy and Calculating Regret

Tayyip Altan

September 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.

In addition to the concepts mentioned above, we will use more pipe operations and apply data subsetting. If you'd like additional material and practice with these techniques before continuing the tutorial, extra exercises are available in the GitHub repository. You can find them in the markdown file 'SP_questions,' with the corresponding answers in 'SP_questions_ANSWERS'.

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.

```r
# Packes required for subsequent analysis. P_load ensures these will be installed and loaded.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(dplyr,
               tidyr,
               ggplot2,
               reshape2,
               latex2exp
               )

# If you do not have tinytex installed, uncomment the command below for knitting the pdf
#tinytex::install_tinytex()
```

Dataset

We will explore the concepts from this problem set with a dataset from Yahoo!. This dataset contains information on 10 articles, detailing when each article was shown to a user and whether the user clicked on it. The data is from a single day and contains +40k observations. If your laptop cannot handle the size of this dataset, we have provided a smaller version with fewer observations for you to use. Please note that results may vary depending on which dataset you use. The dataset contains three columns:

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

# Create an index column
dfYahoo_for_sim$index <- 1:nrow(dfYahoo_for_sim)
```

ϵ-Greedy Algorithm: notation

First, a brief recap of key notation from the lecture that will be useful for this tutorial.

With a denoting an arm, $a \in \mathcal{A}$ indicates that the arm a is one of the options available to our algorithm from the set of all available arms, $\mathcal{A}$. We define a_t as the chosen arm at time t. Our goal is to select arms in a way that gives us the highest possible reward, $Q_t(a)$. In the case of an ϵ-Greedy Algorithm, we pursue the greedy option in $(1-\epsilon)$% of cases, and in ϵ% 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:

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

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

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 is 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 (1).

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 the 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 : This function picks an arm, based on the greedy algorithm for multi-armed bandits
#
# The policy_greedy function takes the following three arguments as input :
#  df: this is a data.frame with three columns: arm, reward, and index
#  eps : a float, which is the percentage of times a random arm needs to be picked
#  n_arms: an integer, which is the total number of arms available. Default value is set to 10.
#
# The function outputs/returns :
# chosen_arm ; integer, index of the arm chosen
####
policy_greedy <- function(df, eps, n_arms=10){

  # Draws a random float between 0 and 1 from a uniform distribution
  random_uniform_variable <- runif(1, min=0, max=1)

  # in epsilon % of cases, pick a random arm (exploration)
  if (random_uniform_variable < eps){

    # TODO: sample uniform random arm between 1:n_arms
    chosen_arm <-


  }

  # in (1-epsilon)% of cases, pick the arm that has the highest average reward
  else{

    # TODO: finish the code below to create the dataframe df_reward_overview
    # this dataframe shows per arm
    # - reward_size: total reward for each arm
    # - sample_size: total observations for each arm
    # - succes_rate: total reward/total observations per arm

    df_reward_overview <- df %>%
      drop_na() %>%
      group_by(arm) %>% # Your code here...
    # TODO: pick the arm with the highest success_rate from df_reward_overview
    chosen_arm <-


  }

  # 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: data.frame with column names "arm", "reward", and "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.
#               The default value is set to 1.
#
#
#   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_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: apply epsilon-greedy algorithm, updating at interval
for (i in 1:n_sim){

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

    # TODO: select the arm with your policy_greedy function and define the current arm
    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(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 ]
```

```r
    # 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

    # update the data.frame df_results_at_t to incorporate the new information
    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 3: simulate the epsilon greedy algorithm

Using the 'sim_greedy' function, run the epsilon greedy algorithm for the following parameters: $n_{\text{before simulation}} = 10$, $n_{simulation} = 100$, $\epsilon = 0.1$ and $\epsilon = 0.25$. Calculate the total reward based on the returned 'df_results_of_policy'.

```r
# set parameters: observations before start simulation, then how many observations for simulation
n_before_sim <- 20
n_sim <- 200

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

# Use the sim_greedy function created in task 2 to simulate the results for epsilon = 0.1
result_sim_eps_01 <- sim_greedy(dfYahoo_for_sim,
                                n_before_sim=n_before_sim,
                                n_sim=n_sim,
                                epsilon=0.1,
                                interval=1)

# TODO: use the sim_greedy function created in task 2 to simulate the results for epsilon = 0.25
# assign the results to a variable called result_sim_eps_025
result_sim_eps_025 <-
```

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?

```r
# create a dataframe with the results per epsilon.
# calculate the cumulative sum over time
df_result_sims <- data.frame(t = 1:n_sim,
                             eps_01 = cumsum(result_sim_eps_01$df_results_of_policy$reward),
                             eps_025 = cumsum(result_sim_eps_025$df_results_of_policy$reward))

# melt the dataframe for ggplot visualization
df_result_sims_melted <- melt(df_result_sims, id = 't')

# TODO: Plot the reward in a line graph to compare performance for epsilon = 0.1 and epsilon = 0.25
```

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

Calculating regret

Regret is the opportunity loss given what we could have received, and what we did receive. The optimal value, denoted by V^*, is the highest possible average reward we can achieve by selecting the best arm from the set of all arms. The total regret at time t is then:

$$L_t = E[\sum_{\tau=1}^{t} V^* - Q(a_\tau)] \tag{2}$$

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

Task 5: calculate regret for $\epsilon = 0.1$

Based on equation (2), calculate the regret for the greedy algorithm when $\epsilon = 0.1$.

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

# total reward for our algorithm
total_reward_eps_01 <- tail(df_result_sims$eps_01, n=1)

# obtain the average reward per arm
df_result_per_arm_sample <- result_sim_eps_01$df_sample_of_policy %>%
  group_by(arm) %>%
  summarise(avg_reward = mean(reward))

# TODO: calculate regret for the greedy algorithm when epsilon = 0.1 by using the following steps:
#   1. Select the average reward of the best arm
#   2. Determine the total reward of the best arm
#   3. Calculate the regret
```