08 - Multi-Armed Bandit

frequentist bayesian

In this exercise session we will consider the UCB1 and TS algorithms as examples of frequentist and Bayesian MAB algorithms, respectively, for the solutions of the stochastic MAB problem.

Let us instantiate a stochastic MAB environment with 5 arms with Bernoulli reward:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

#Set the environment
exp_reward = np.array([0.15, 0.1, 0.1, 0.2, 0.35, 0.2])
n_arms = len(exp_reward)
opt = np.max(exp_reward)
idx_opt = np.argmax(exp_reward)
deltas = opt - exp_reward
deltas = np.array([delta for delta in deltas if delta > 0])

#Time horizon
T = 5000
```

we have 5 arms and for each arm we have a Bernoulli distribution (the parameters are the ones written)

maximum reward among the expected

index of the optimal expected reward (= index of the best arm)

Recall that in this example we are aware of the environment, while in real-world applications we do not have the model of the environment, therefore selecting the optimal arm is not a viable operation.

Instead, we want to resort to an online approach to minimize the loss due to the learning of the optimal arm.

UCB1

As a first algorithm we will implement the UCB1 algorithm, which uses **upper confidence bounds** to select the arm to pull for the next round.

we have an empirical estimate of the mean of the rewards, moreover we can obtain an information on the uncertainty of this estimate → the algorithm uses these infos to build an upper confidence bound and to choose the arm that have the higher upper confidence bound

We recap here the main steps of the algorithm:

- For each time step t
  - Compute $\hat{R}_t(a_i) = \frac{\sum_{i=1}^{t} r_{i,t}\, \mathbf{1}\{a_i = a_{i_t}\}}{N_t(a_i)} \quad \forall a_i$ — for each arm compute the sample mean of the rewards
  - Compute $B_t(a_i) = \sqrt{\frac{2 \log t}{N_t(a_i)}} \quad \forall a_i$ — confidence bound for each arm
- Play arm $a_{i_t} = \arg\max_{a_i \in \mathcal{A}} \left(\hat{R}_t(a_i) + B_t(a_i) \right)$ — estimate + confidence

After the execution of the algorithm we would like to evaluate its performances. In this case, as usual in the MAB literature, we resort to the concept of expected pseudo-regret:

$$L_T = TR^* - \mathbb{E}\left[\sum_{t=1}^{T} R(a_{i_t})\right] = \sum_{a \in \mathcal{A}} \mathbb{E}[N_T(a_i)]\Delta_i$$

what we could've obtain in expectation choosing always the best arm

what we obtained in expectation using our algorithm

A first approximated version of the regret can be computed as follows:

$$\tilde{L}_T = \sum_{t=1}^{T} r(a^*) - r(a_{i_t})$$

optimal reward − reward obtained with our choice

Let's see how it can be implemented:

In [2]:
```
def UCB1():
  ucb1_criterion = np.zeros(n_arms)
  expected_payoffs = np.zeros(n_arms)
  number_of_pulls = np.zeros(n_arms)

  regret = np.array([])
  pseudo_regret = np.array([])

  for t in range(1, T+1):
      #Select an arm
      if t < n_arms:
        pulled_arm = t  # round robin for the first n steps
      else:
        idxs = np.argwhere(ucb1_criterion == ucb1_criterion.max()).reshape(-1)  # there can be more arms
        pulled_arm = np.random.choice(idxs)

      #Pull an arm
      reward = np.random.binomial(1, exp_reward[pulled_arm])
      if pulled_arm != idx_opt:
        reward_opt = np.random.binomial(1, exp_reward[idx_opt])
      else:
        reward_opt = reward

      #Update UCB1
      number_of_pulls[pulled_arm] = number_of_pulls[pulled_arm] + 1
      expected_payoffs[pulled_arm] = ((expected_payoffs[pulled_arm] * (number_of_pulls[pulled_arm] - 1.0) + reward) /
                                      number_of_pulls[pulled_arm]) # update sample mean for the selected arm
      for k in range(0, n_arms):
        ucb1_criterion[k] = expected_payoffs[k] + np.sqrt(2 * np.log(t) / (number_of_pulls[k] + 1))

      #Store results
      regret = np.append(regret, reward_opt - reward)
```

we'll store the upper bounds

we'll store the empirical means of the rewards

for each arm we'll store how many times we pulled the arm

approximated regret

pseudo regret that we can compute exactly only because we know all the values

we want to select each arm at least once

we take at random one of the arms that have ucb1 max

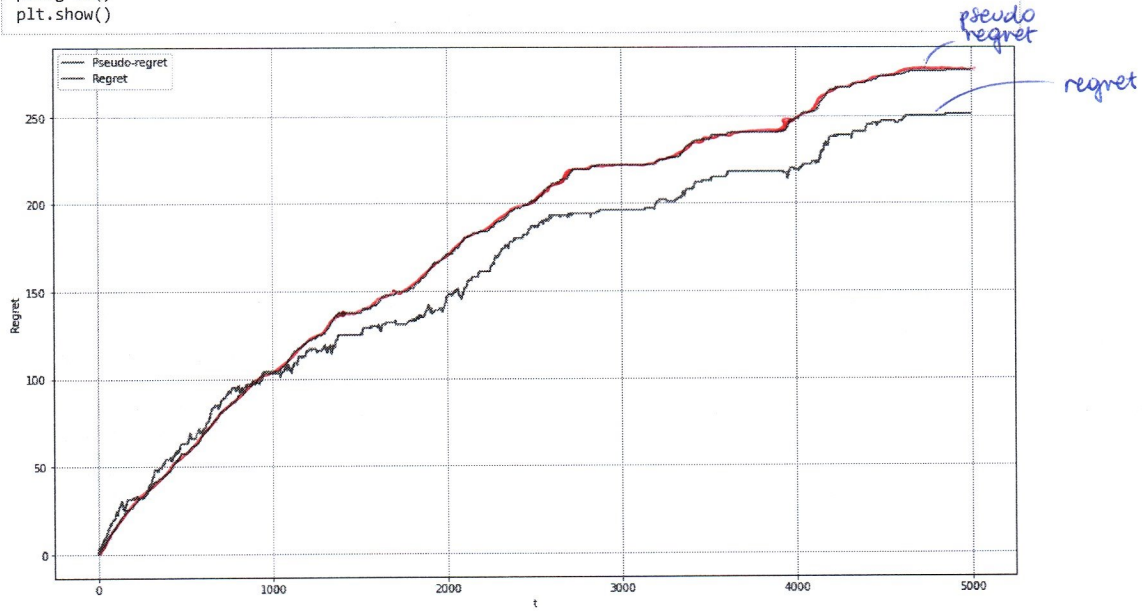
```
        pseudo_regret = np.append(pseudo_regret, opt - exp_reward[pulled_arm])
    return regret, pseudo_regret
```

In [3]:
```
regret, pseudo_regret = UCB1()
```

An example of the execution of the UCB1 algorithm is available at:
https://drive.google.com/file/d/1OMDBpbvCldsXdnBnnXGZIoZNDfchHoj1/view?usp=sharing

Here we compare the approximated regret with the expected pseudo-regret we obtained with a single run:

In [4]:
```
plt.figure(figsize=(16,9))
plt.ylabel("Regret")
plt.xlabel("t")
plt.plot(np.cumsum(pseudo_regret), color='r', label='Pseudo-regret')
plt.plot(np.cumsum(regret), color='g', label='Regret')
plt.legend()
plt.grid()
plt.show()
```



Notice that the expected pseudo-regret is monotonically increasing over time, and, conversely, the regret might also decrease.

However, to evaluate the performance of our approach in the correct way we also have to average between multiple runs, in order to take into account the of the environment randomness. To evalate the uncertainty of the estimates, we also add the 95% confidence intervals.

In [5]:
```
n_repetitions = 5
regrets, pseudo_regrets = np.zeros((n_repetitions, T)), np.zeros((n_repetitions, T))
for i in range(n_repetitions):
    regrets[i], pseudo_regrets[i] = UCB1()
```

In [6]:
```
# Compute the cumulative sum
cumu_regret = np.cumsum(regrets, axis=1)
cumu_pseudo_regret = np.cumsum(pseudo_regrets, axis=1)

# Take the average over different runs
avg_cumu_regret = np.mean(cumu_regret, axis=0)
avg_cumu_pseudo_regret = np.mean(cumu_pseudo_regret, axis=0)

std_cumu_regret = np.std(cumu_regret, axis=0)
std_cumu_pseudo_regret = np.std(cumu_pseudo_regret, axis=0)
```

We also want to understand wheter we are respecting the UCB1 upper bound:

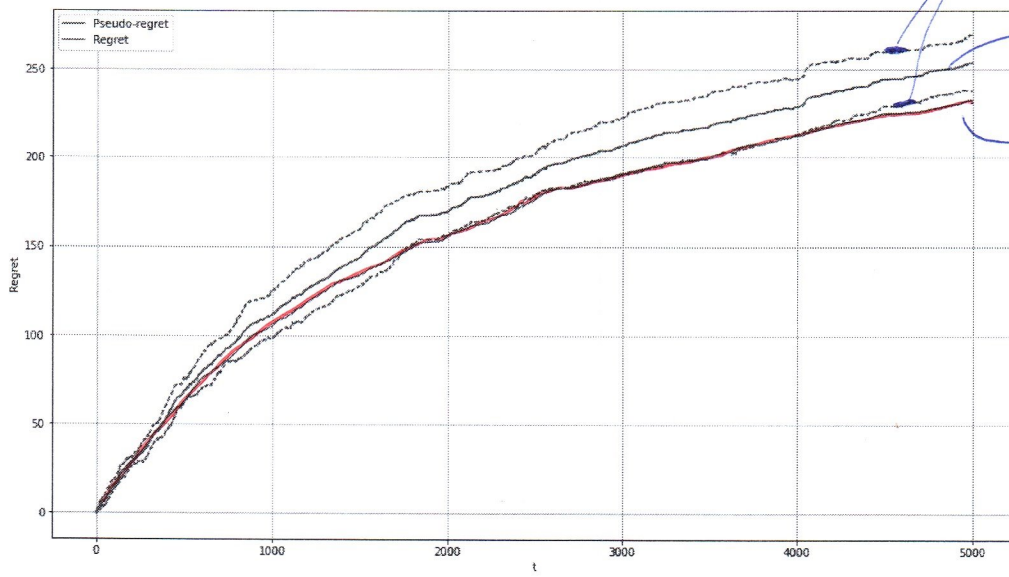
$$L_T \leq 8 \log T \sum_{i|\Delta_i>0} \frac{1}{\Delta_i} + \left(1 + \frac{\pi^2}{3}\right) \sum_{i|\Delta_i>0} \Delta_i$$

In [7]:
```
ucb1_upper_bound = np.array([8*np.log(t)*sum(1/deltas) + (1 + np.pi**2/3)*sum(deltas)
                             for t in range(1,T+1)])
```

In [8]:
```
plt.figure(figsize=(16,9))
plt.ylabel("Regret")
plt.xlabel("t")
plt.plot(avg_cumu_pseudo_regret, color='r', label='Pseudo-regret')

plt.plot(avg_cumu_regret + 1.96 * std_cumu_regret / np.sqrt(n_repetitions), linestyle='--', color='g')
plt.plot(avg_cumu_regret, color='g', label='Regret')
plt.plot(avg_cumu_regret - 1.96 * std_cumu_regret / np.sqrt(n_repetitions), linestyle='--', color='g')
#plt.plot(ucb1_upper_bound, color='b', label='upper bound')

plt.legend()
plt.grid()
plt.show()
```



confidence interval

regret (average on different runs)

pseudo regret

Thompson Sampling

In this second example we are resorting to a Bayesian method. For each arm we use a uniform distribution as prior for the expected value of the reward, i.e., $\mu_i \sim Beta(1,1) = U([0,1])$. During the learning process we update the beta distribution using the posterior provided by the reward:

- if we observe a success: we update the α parameter of the Beta increasing its value by one
- if we observe a failure: we update the β parameter of the Beta increasing its value by one

Formally:

THOMSON SAMPLING

- consider a bayesian prior for each arm: $f_1, \dots, f_N$
- at each round t sample from each one of the distributions: $\hat{r}_1, \dots, \hat{r}_N$
- pull the arm a_{i_t} with the highest sampled value: $i_t = \arg\max_i \hat{r}_i$
- update the priors incorporating new informations (actually we update only the prior of the arm we pulled)

In [9]:
```
def thompson_sampling():
  T = 5000
  beta_parameters = np.ones((n_arms,2))

  regret = np.array([])
  pseudo_regret = np.array([])

  for t in range(1,T+1):
      #Select an arm
      samples = np.random.beta(beta_parameters[:,0], beta_parameters[:,1])
      pulled_arm =  np.argmax(samples)

      #Pull an arm
      reward = np.random.binomial(1, exp_reward[pulled_arm])
      if pulled_arm != idx_opt:
        reward_opt = np.random.binomial(1, exp_reward[idx_opt])
      else:
        reward_opt = reward

      #Update TS
      beta_parameters[pulled_arm,0] =  beta_parameters[pulled_arm,0] + reward
      beta_parameters[pulled_arm,1] =  beta_parameters[pulled_arm,1] + 1.0 - reward

      #Store results
      regret = np.append(regret, reward_opt - reward)
      pseudo_regret = np.append(pseudo_regret, opt - exp_reward[pulled_arm])
  return regret, pseudo_regret, beta_parameters
```

As before, we start analysing the single run

In [10]:
```
regret, pseudo_regret, beta_parameters = thompson_sampling()
```

In [11]:
```
print(beta_parameters)
```

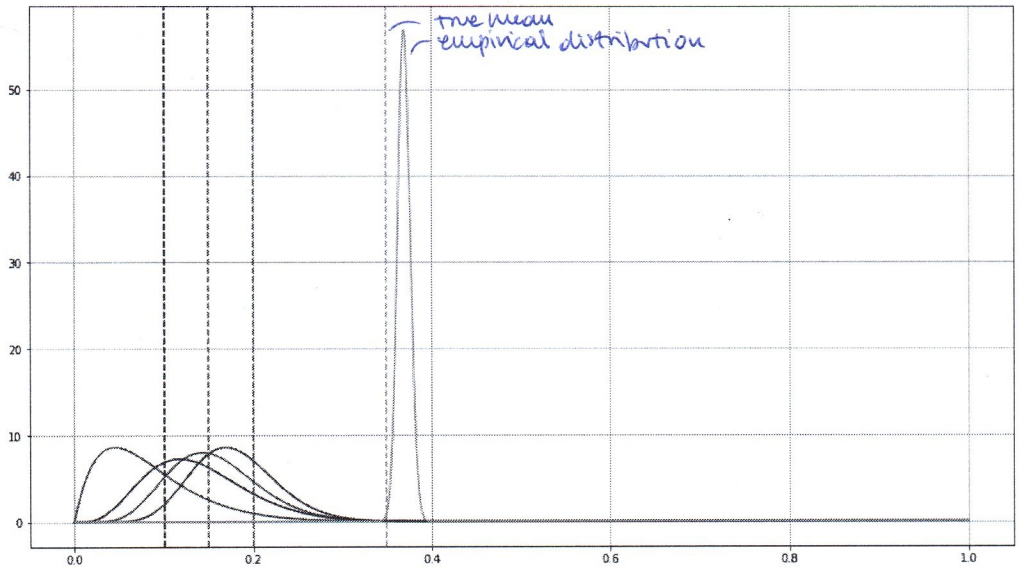
```
[[8.000e+00 4.300e+01]
 [2.000e+00 2.200e+01]
 [5.000e+00 3.100e+01]
 [1.200e+01 5.500e+01]
 [1.752e+03 2.988e+03]
 [2.000e+01 7.400e+01]]
```

Notice how the parameters for the arms with larger expected rewards have larger values, meaning that during the learning process we selected those arms more often than the others.

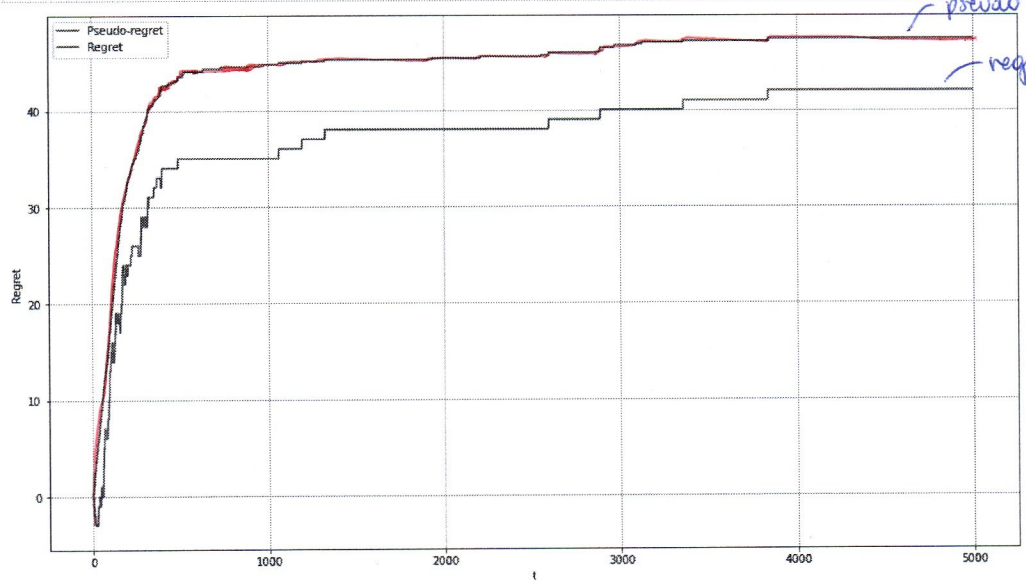
An example of the execution of the TS algorithm is available at: https://drive.google.com/file/d/1CC1YVM06-AEWQ71dXxoZPNBDUZpjeWSo/view?usp=sharing

In [12]:
```
from scipy.stats import beta
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 9))
x = np.linspace(0,1,1000)
colors = ['red', 'green', 'blue', 'purple', 'orange']
for params, rew, color in zip(beta_parameters, exp_reward, colors):
  rv = beta(*params)
  plt.plot(x, rv.pdf(x), color=color)
```

```
  plt.axvline(rew, linestyle='--', color=color)
plt.grid()
plt.show()
```



In [13]:
```
plt.figure(figsize=(16,9))
plt.ylabel("Regret")
plt.xlabel("t")
plt.plot(np.cumsum(pseudo_regret), label='Pseudo-regret', color='r')
plt.plot(np.cumsum(regret), label='Regret', color='g')
plt.legend()
plt.grid()
plt.show()
```



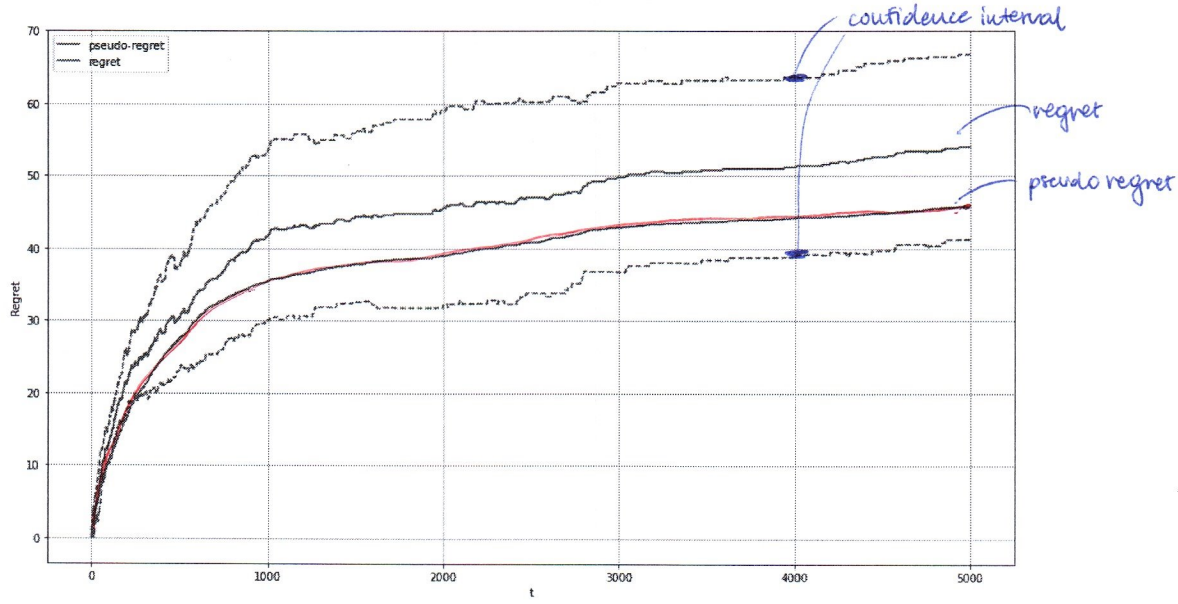
In [14]:
```
n_repetitions = 5
regrets, pseudo_regrets = np.zeros((n_repetitions, T)), np.zeros((n_repetitions, T))
for i in range(n_repetitions):
  regrets[i], pseudo_regrets[i], _ = thompson_sampling()

# Compute the cumulative sum
cumu_regret = np.cumsum(regrets, axis=1)
cumu_pseudo_regret = np.cumsum(pseudo_regrets, axis=1)

# Take the average over different runs
avg_cumu_regret = np.mean(cumu_regret, axis=0)
avg_cumu_pseudo_regret = np.mean(cumu_pseudo_regret, axis=0)

std_cumu_regret = np.std(cumu_regret, axis=0)
std_cumu_pseudo_regret = np.std(cumu_pseudo_regret, axis=0)
```

In [15]:
```
plt.figure(figsize=(16,9))
plt.ylabel("Regret")
plt.xlabel("t")
plt.plot(avg_cumu_pseudo_regret, color='r', label='pseudo-regret')
plt.plot(avg_cumu_regret + std_cumu_regret, linestyle='--', color='g')
plt.plot(avg_cumu_regret, color='g', label='regret')
plt.plot(avg_cumu_regret - std_cumu_regret, linestyle='--', color='g')
plt.legend()
plt.grid()
plt.show()
```



Homeworks

MAB Lower bound

Visualize the regret Lower bound for stochastic MAB problems. Check if there exists a Lower bound which does not requires to know the distribution, i.e., that does not require to know the arms' gaps Δ_i and visualize it.

In [16]:
```
# TODO
```

Thompson Sampling Upper Bound

Visualize the pseudo regret and the Upper bound for the considered problem provided by TS algorithm.

In [17]:
```
# TODO
```

Adversarial MAB

Model an adversarial MAB environment with 5 arms and $\{0, 1\}$ rewards. A suggestion is to consider an adversary which decides the rewards for all the rounds in advance.

In [18]:
```
# TODO
```

EXP3

Implement the EXP3 algorithm for adversarial bandit defined in the MAB setting defined before. Compute the expected regret and compare it with the theoretical upper and lower bounds.

In [19]:
```
# TODO
```

8 Multi-Armed Bandit

8.1 Questions

✗ Exercise 8.1

Tell if the following statements about MAB are true or false and provide motivations for your choice.

1. The MDP corresponding to a MAB setting has no state.
2. The MDP corresponding to a MAB setting has no transition probabilities.
3. An algorithm which always uses the greedy choice (maximize the empirical expected reward) might get stuck to suboptimal solutions.
4. We are able to solve the exploration/exploitation problem by considering either upper or lower bounds on the expected rewards.
5. In a frequentist framework, if we base our sequential policy on a MAB by considering only $\hat{R}(a_i)$ we are not using all the information we have about the estimates.
6. Uncertainty over quantities is usually handled by explorative choices in the MAB setting.

✗ Exercise 8.2

Tell if the following problems can be modeled as MAB and explain why.

1. Maze escape;
2. Pricing goods with stock;
3. Pricing goods without stock;
4. Optimal bandwidth allocation (send the largest amount of information without congesting the band);
5. Web ads placement;

6. Weather prediction (with multiple experts).

If they fit the MAB setting, is the environment adversarial or stochastic?

Exercise 8.3

Provide an example for which the pure exploitation strategy is failing to converge to the optimum in a MAB setting with Bernoulli rewards. Recall that the pure exploitation algorithm chooses the arm by selecting the one with the largest:

$$\hat{R}_t(a_i) = \frac{1}{N_t(a_i)} \sum_{j=1}^{t} r_{i,j} \, \mathbb{I}\left\{a_i = a_{i_j}\right\},$$

where $\mathbb{I}$ is the indicator function.

How often does this occurs? Can we compute an lower bound over the regret of this strategy?

✗ Exercise 8.4

The ε-greedy algorithm selects the best action except for small percentages of times ε, where all the actions are considered. Consider a MAB stochastic setting.

1. Is this algorithm converging to the optimal strategy (in some sense)?
2. If not, propose a scheme which has the chance of converging to the optimal solution.
3. Are we in a MAB perspective if we are using this algorithm?

MAB is a semplification of RL case, with just 1 state, but also we have something different. In RL we want to find the optimal solution, in MAB we want also to understand how much we're losing during time (how much regret we're accumulating)

With the ε-greedy strategy we're just trying to explore enough to get the optimal solution, hence we're not in a really MAB perspective.

Exercise 8.5

Write the formula for the minimum regret we might have on average over $T = \lceil e^{10} \rceil$ time steps in the case we have a stochastic MAB with 3 arms and expected rewards:

$$R(a_1) = 0.2 \tag{8.1}$$
$$R(a_2) = 0.4 \tag{8.2}$$
$$R(a_3) = 0.7 \tag{8.3}$$

and each distribution $\mathcal{R}(a_i)$ is Bernoulli.

Note that the KL divergence for Bernoulli variables with means p and q is:

$$KL(p, q) = p \log\left(\frac{p}{q}\right) + (1 - p) \log\left(\frac{(1 - p)}{(1 - q)}\right).$$

Hints: $\log(\frac{0.2}{0.7}) = -1.25$, $\log(\frac{0.8}{0.3}) = 0.98$, $\log(\frac{0.4}{0.7}) = -0.56$ and $\log(\frac{0.6}{0.3}) = 0.69$.

Is it possible that your algorithm achieves lower regret? If so, provide an example.

Exercise 8.6

Provide examples of either Bayesian or frequentist MAB algorithm showing the following properties:

1. It incorporates expert knowledge about the problem in the arms;
2. It provides tight theoretical lower bound on the expected regret in the stochastic setting;
3. It provides tight theoretical upper bound on the expected regret in the stochastic setting;
4. At each turn, it modifies only the statistics of the chosen arm.

Motivate your answers.

✗ Exercise 8.7

Consider the following bounds (supposed to hold with probability at least δ) for a MAB stochastic setting:

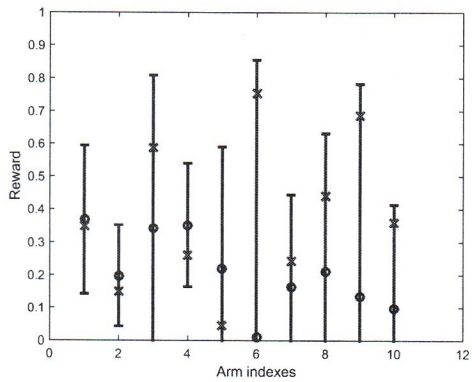


Figure 8.1: Bounds in a MAB with $N = 10$.

where the blue bars are the Hoeffding bounds for the expected rewards, the blue circles are the estimated expected rewards, and the red crosses are the real expected rewards.

1. Which arm would a UCB1 algorithm choose for the next round?

2. Do you think that Figure 8.1 might be the results obtained by running UCB1 for several rounds?
3. Which arm will UCB1 converge to if $T \to \infty$?
4. Which arm is the one which we pulled the most so far?

Motivate your answers.

Exercise 8.8

Consider the Thompson Sampling algorithm. Assume to have the following posterior distributions $Beta_i(\alpha_t, \beta_t)$ for arms $\mathcal{A} = \{a_1, \ldots, a_5\}$ rewards, which are distributed as Bernoulli r.v.:

$$
\begin{aligned}
a_1 &: \quad \alpha_t = 1 \quad \beta_t = 5 \\
a_2 &: \quad \alpha_t = 6 \quad \beta_t = 4 \\
a_3 &: \quad \alpha_t = 11 \quad \beta_t = 23 \\
a_4 &: \quad \alpha_t = 12 \quad \beta_t = 33 \\
a_5 &: \quad \alpha_t = 28 \quad \beta_t = 21
\end{aligned}
$$

From these distribution you extract the following samples for the current round:

$$
\begin{aligned}
\hat{r}(a_1) &= 0.63 \\
\hat{r}(a_2) &= 0.35 \\
\hat{r}(a_3) &= 0.16 \\
\hat{r}(a_4) &= 0.22 \\
\hat{r}(a_5) &= 0.7
\end{aligned}
$$

1. Which arm would the TS algorithm play for the next round?
2. What changes if the real distributions of the arm rewards are Gaussian.
3. Assume we started the TS algorithm with uniform $Beta(1, 1)$ priors. What would UCB1 have chosen in the case of Bernoulli rewards for the next round?

Solutions

Answer of exercise 8.1

1. FALSE It has a single state. There is no dynamics: we stay in the same state if action we take
2. FALSE The transition probability matrix is $P(s, a_i|s) = 1$.
3. TRUE There are cases in which if we only consider the expected values of the rewards as decision tool, we might discard good options even if we only gathered small evidence from them.
4. FALSE Pessimistic choices (lower bounds) would not solve the exploration/exploitation dilemma since, this way, we do not provide incentives to explore the options which are considered suboptimal so far.
5. TRUE We are also allowed to consider concentration inequalities which provides us information about the uncertainty we have about the reward estimate.
6. TRUE Sometimes, we want to reduce uncertainty by picking suboptimal choices which allow us to gather more information that will be beneficial in the future.

Answer of exercise 8.2

1. NO There are plenty of states in a maze escape problem (the reward cannot be modeled as a single distribution).
2. NO We are limited in the number of successes, thus we might stop the problem at some point (it can be a generalized MAB, in the specific a budget MAB). (again we have multiple states)
3. YES Rewards are stochastic if we consider a distribution of users or adversarial if we assume to have a single malicious buyer.
4. YES If we consider the traffic in the bandwidth as a stochastic event.
5. YES Assuming, for instance, a pay-per-click scheme and a click event which is a stochastic event.
6. NO Each time we predict we have rewards coming from all the arms (expert problem). If we use only the feedback coming from the pulled arm, we can use the MAB model, even if we would discard some useful information by doing this.

Answer of exercise 8.3

The most simple example is the one in which the first time we pull the optimal arm

we have a zero reward. This occurs with probability $1 - \mu^*$ and thus, being positive probability that the arm is pulled only once, the regret cannot be less than linear in the time horizon, i.e., $O(T)$.

Answer of exercise 8.4

There will always be a small percentage of times that we go random (random among the non-optimals)

1. No, it can not reach the optimal policy since it will always select the suboptimal arms with a fixed probability. This leads to a regret of the order $\varepsilon T \gg O(\log T)$.
2. If we use a strategy s.t. $\lim_{t\to\infty} \varepsilon(t) = 0$ we might converge.
3. Even if this strategy converges, you do not have any assurance about the regret we are suffering in the process of learning. There exists a theoretical analysis of a version of the ε-greedy algorithm in:
   Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multi-armed bandit problem." Machine learning 47.2-3 (2002): 235–256
   The solution proposed can be applied only by knowing the gaps Δ_i between arms, otherwise the bound might not hold.

Answer of exercise 8.5

If we assume that T is sufficiently large, we have:

$$R_T \geq \log T \sum_{a_i \neq a^*} \frac{\Delta_i}{KL(\mathcal{R}(a_i), \mathcal{R}(a^*))},$$

thus in our case:

$$\begin{aligned} R_T &\geq \log e^{10} \left(\frac{0.7 - 0.2}{KL(0.2, 0.7)} + \frac{0.7 - 0.4}{KL(0.4, 0.7)} \right) \\ &= 10 \left(\frac{0.5}{0.2(-1.25) + 0.8(0.98)} + \frac{0.3}{0.4(-0.56) + 0.6(0.69)} \right) \\ &= 25.1528 \end{aligned}$$

We might violate our lower bound in some cases, e.g.:

- if we consider a subset of MAB settings;
- if we consider a single realization of the rewards.

Answer of exercise 8.6

1. TS As usual we can use prior distributions to incorporate information about the problem.

2. NONE The lower bound is defined over the problem and not by basing on the algorithm we used.

3. UCB1 and TS Provide order optimal theoretical upper bound on the expected regret in the stochastic setting. TS matches also the constant (KL-UCB has the same assurance).

4. TS It only modifies the Beta distribution corresponding to the pulled arm.

Answer of exercise 8.7

1. a_6 since it is the one with the highest upper bound.

2. No, since in principle UCB1 keeps all the upper bounds at similar level.

3. a_6, since it is the one with highest expected value.

4. a_2, since it is the one with smallest bounds (which are inversely proportional to the number of pulls).

Answer of exercise 8.8

1. Thompson sampling will choose the arm a_i with the largest sampled $\hat{r}(a_i)$, which in this case is a_5.

2. In the case we are considering Gaussian rewards it is not meaningful to use Beta distribution as prior. For instance, if the reward are Gaussian they might provide also negative values, while the support of the Beta is $[0, 1]$. Thus, it makes no sense to instantiate a problem with Beta prior when you have Gaussian rewards.

3. Since we started with uniform prior and at each round we collected a success or a failure we are currently at round:

$$t = 4 + 8 + 32 + 43 + 47 = 134,$$

while the UCB1 upper bound $U_t(a_i)$ is of the form:

$$U_t(a_i) = \frac{\alpha_t - 1}{\alpha_t + \beta_t - 2} + \sqrt{\frac{2 \log t}{\alpha_t + \beta_t - 2}},$$

thus:

$$U_t(a_1) = \frac{0}{4} + \sqrt{\frac{2\log 134}{4}}$$
$$U_t(a_2) = \frac{5}{8} + \sqrt{\frac{2\log 134}{8}}$$
$$U_t(a_3) = \frac{10}{32} + \sqrt{\frac{2\log 134}{32}}$$
$$U_t(a_4) = \frac{11}{43} + \sqrt{\frac{2\log 134}{43}}$$
$$U_t(a_5) = \frac{27}{47} + \sqrt{\frac{2\log 134}{47}}$$

In this case the UCB1 algorithm chooses the arm a_i s.t. $i = \arg\max_i U_t(a_i)$, in this case a_2.