Machine Learning

Notes Part 5, HSG-MiQEF, Spring 2022

David Preinerstorfer

Last updated on 2022-03-23 15:40:22

Contents

Goals:

In this lecture, we briefly consider the class of problems which are summarized as reinforcement learning, and take a more detailed look at the problem of **multi-armed bandits**.

For a detailed and accessible treatment of multi-armed bandits the reader is recommended to take a look at the great textbook Lattimore and Szepesvári [2020], and also the summary paper Bubeck and Cesa-Bianchi [2012]. Furthermore, the paper Charpentier et al. [2021] could constitute an interesting/quick read and is recommended.

1 Reinforcement learning

In contrast to the supervised learning setup we considered up to now, and where we always exploited a training sample to learn optimal decision rules (which we characterized in the first part of these lecture notes), we are now taking a look at a situation, where an **agent gradually learns by interacting with an environment**. In particular, there is a specific goal the agent wants to achieve, but the problem is that the agent is uncertain about the environment.

In a general reinforcement-learning setup an agent needs to choose among actions. The actions chosen may have an impact on the state of the agent. The effects of the actions of the agent on the state cannot be predicted with certainty. Examples would be moves in games such as chess, autonomous cars, online advertising, etc.

The general idea is that from interacting with the environment the agent gradually learns the effect of the possible actions on the state and exploits this knowledge. Doing so, the agent tries to **explore** the effect of all possible actions, whereas one would also like to **exploit** what one has already learned. This is called **exploration-exploitation-trade-off** and will become clear in the examples below.

In this lecture, we will mainly look at multi-armed bandits, which (due to its simplicity) are quite easy to comprehend (while being not so easy to analyse theoretically, due to the sequential situation). Another popular problem in reinforcement learning is called Q-learning. This is a special type of a Markovian decision problem, which leads to Bellman equations, which can be solved through value function iteration. This should sound familiar from your macroeconomics courses. Therefore, we rather look at multi-armed bandits, as they are potentially unfamiliar to you.

2 Multi-armed bandits

2.1 Problem setup

In the simplest instance of a multi-armed bandit problem, at each point in time $t = 1, 2, \ldots$, the agent needs to decide which among a finite number of actions (**arms**) $i = 1, \ldots, K$ to take (**pull**). Given that the agent has decided to pull arm i, the agent receives a random reward $Y_{i,t}$, say. Which arm the agent wants to pull can be decided by the agent based on all previous outcomes and assignments that have been observed until time t (the agent therefore can learn from previous results, and may adapt his choices to the aggregated information).

The underlying **probabilistic model** (note that there are also non-probabilistic results on mulit-armed

bandits, which we shall not go into detail) is a sequence of random vectors $\boldsymbol{Y}_t$, say, of which at each point in time t one can only observe a single coordinate. The coordinates that one does not observe (i.e., the arms one does not pull) remain unobserved; in the terminology of causal inference, these random variables are **counterfactuals** and the vector $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{K,t})'$ is the vector of **potential outcomes** (of which one can only observe a single coordinate). Throughout (and in correspondence with the literature on multi-armed bandits) the sequence $\boldsymbol{Y}_t$ is assumed to be i.i.d., but the distribution of the random vector $\boldsymbol{Y}_t$ is unknown, and often left completely unspecified beyond support conditions.

The arm pulled by the agent at time t is denoted by

$$\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1}),$$

where $\boldsymbol{Z}_{t-1}$ contains all outcomes and $\boldsymbol{A}_{t-1}$ contains all assignment decisions that have been observed in previous rounds, i.e.,

$$\boldsymbol{Z}_{t-1} = (Y_{\pi_{t-1}(\boldsymbol{Z}_{t-2}, \boldsymbol{A}_{t-2}),t-1}, Y_{\pi_{t-2}(\boldsymbol{Z}_{t-3}, \boldsymbol{A}_{t-3}),t-2}, \ldots) \quad \text{and} \quad \boldsymbol{A}_{t-1} = (\pi_{t-1}(\boldsymbol{Z}_{t-2}, \boldsymbol{A}_{t-2}), \pi_{t-2}(\boldsymbol{Z}_{t-3}, \boldsymbol{A}_{t-3}), \ldots).$$

In principle, the decision maker could also incorporate an external random mechanism, e.g., a coin flip (breaking ties, for example). We allow implicitly for such an additional source of randomness (that is independent of the sequence $\boldsymbol{Y}_t$), without incorporating that into our notation. Writing $\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1})$ is a bit clumsy, but it highlights the difference between $\pi_t : \mathbb{R}^{t-1} \times \mathbb{R}^{t-1} \to \{1, \ldots, K\}$, which is a function the decision maker applies to previous outcomes and assignments to decide which arm to pull, and the previous information available, which is summarized in $\boldsymbol{Z}_{t-1}$ and $\boldsymbol{A}_{t-1}$, and which one plugs into π_t to obtain the arm to pull $\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1})$.

2.2 Policies

Explicitly or implicitly, the agent decides on functions π_t to use. A sequence of such functions $\pi_1, \pi_2, \ldots$ is called a **policy**, and we write $\boldsymbol{\pi}$ for such a sequence. [There is one thing which we do not highlight in our notation. In principle, the functions π_t could also incorporate the horizon n, i.e., if the agent knows that he will pull n times in total, the agent might incorporate that knowledge into the policy (or may not). If the agent does not, the agent uses an **anytime** policy. Otherwise, the policy is not anytime, and uses more information. We will only take a look at anytime policies here.]

A first (very) naive policy could be the following one: in words, just pull the arms $i = 1, \ldots, K$ cyclically, starting with the first arm. That is, start with arm 1, then arm 2 and so one. Once all arms have been assigned once, start again with arm 1 and so on. This policy **explores** all arms equally well, the bad ones

(i.e., the ones with low average returns) as well as the good ones (i.e., the ones with comparably high average returns). The advantage is, that all arms are explored. The disadvantage is that one does *not* **exploit** the information accumulated over time! We summarize the above policy, which we refer to as the **uniform allocation policy** below:

Uniform allocation policy

Assign $\pi_t = (t \bmod K) + 1$.

Here $t \mod K$ denotes the remainder after dividing t by K. So, why is this not a good approach? We first need to say what our goal is, and what the best arm actually is.

Note that in the present setting, we can no longer rely directly on the loss/risk optimality framework we used in supervised learning. The learning task and setting is different. We therefore have to first develop something like a risk function.

2.3 Goals

Recall that the until now somewhat vaguely defined goal was to maximize the reward attained (think of each coordinate of $\boldsymbol{Y}_t$ as a reward). Since we are observing realizations of random variables, and taking expected reward as the target (which is not undisputed in the literature), we would like to pull the arm with the highest average reward all the time. This is impossible, in general, because we do not know which of the K arms realizes that maximum. Denote the expected rewards of the arms by

$$\mu_1, \ldots, \mu_K.$$

Hence, the expected reward when pulling arm i is μ_i. Denote an index corresponding the an arm with highest average return by

$$i^* \in \arg\max_{i=1,\ldots,K} \mu_i;$$

note that the set of maximizers may not be a singleton set.

If we would always pull arm i^*, we would observe $Y_{i^*,1}, Y_{i^*,2}, Y_{i^*,3}, \ldots$. Given a policy $\boldsymbol{\pi}$, instead of pulling arm i^*, we pull $\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1})$ which may not coincide with i^*. Note that the expected difference between $Y_{i^*,t}$ (what we should do) and $Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1}),t}$ (what we actually decided to do) coincides with (conditioning

on $\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1})$)

$$\begin{aligned}\mathbb{E}\left(Y_{i^*,t} - Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1}),t}\right) &= \mu_{i^*} - \mathbb{E}\left(Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1}),t}\right) \\ &= \sum_{i=1}^{K} \left[\mu_{i^*} - \mu_i\right] \mathbb{P}\left(\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1}) = i\right).\end{aligned}$$

If we write $\Delta_i = \mu_{i^*} - \mu_i$ for the difference between the expected outcome of a best arm minus the expected outcome of the i-th arm (which is positive, unless arm i is best as well), we can further write

$$\mathbb{E}\left(Y_{i^*,t} - Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1}),t}\right) = \sum_{i=1}^{K} \Delta_i \mathbb{P}\left(\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1}) = i\right).$$

Since we make a decision at every point in time t, we accumulate the differences $Y_{i^*,t} - Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1})}$ over time $t = 1, 2, \ldots$. Hence, in evaluating policies, we need to take a look at the accumulated expected values in the previous display. That is, the criterion we use to evaluate a policy is the sum

$$R(\boldsymbol{\pi}, n) = \sum_{i=1}^{n} \left(\mathbb{E}\left(Y_{i^*,t} - Y_{\pi_t(\boldsymbol{Z}_{t-1},\boldsymbol{A}_{t-1}),t}\right)\right) = \sum_{t=1}^{n}\sum_{i=1}^{K} \Delta_i \mathbb{P}\left(\pi_t(\boldsymbol{Z}_{t-1}, \boldsymbol{A}_{t-1}) = i\right).$$

This quantity is called accumulated expected **regret** of the policy $\boldsymbol{\pi}$ at time n. One goal in multi-armed bandits is to construct policies $\boldsymbol{\pi}$ with small accumulated expected regret. This is achieved, essentially, by pulling a best arm as often as possible, and by avoiding to pull any inferior arm. The difficulty in achieving this, as alluded to already above, is that we do not know the identity of the best arm and therefore have to exploit that while making (unavoidable) mistakes.

Note that for any policy $\boldsymbol{\pi}$ the *function*

$$n \mapsto R(\boldsymbol{\pi}, n)$$

is non-decreasing. Different policies therefore give rise to different non-decreasing functions. We shall evaluate policies according to the **rate** at which $R(\boldsymbol{\pi}, n)$ increases in n. To interpret this, and see what the worst-case rate is, note that if we always pull a wrong arm, the regret would grow as $n \max_{i=1,\ldots,K} \Delta_i$, i.e., would grow linearly in n! Hence, a linear growth of the accumulated expected regret is the **worst case growth rate** behavior!

In the context of this optimality criterion, we can also revisit the accumulated expected regret of the uniform allocation policy introduced above: For that policy it clearly holds that

$$R(\boldsymbol{\pi}, n) = \sum_{t=1}^{n} \Delta_{(t \bmod K)+1}.$$

We therefore see that the accumulated expected regret grows linearly in n (in case there is at least one arm that has an inferior expected regret). Let's illustrate this in a simulation (throughout we shall in the

simulation always look at the situation when there are only 2 arms, i.e., when $K = 2$; furthermore, the test bed will be normally distributed $Y_{i,t}$ with means $\mu_1 = 1/2$ and $\mu_2 = \delta$ or uniformly distributed on $[0, 1/2]$ or $[0, \delta]$, where (in both cases) $\delta \in (0, 1/2)$. Therefore, in both scenarios the second arm is inferior.

Let's first write a function `eval.pol` that evaluates an arbitrary input policy by running the policy `n.rep` times, and where the regret in each of the runs is accumulated and returned. The goal is to develop a benchmark with which we can evaluate different policies that we encounter. The input to `eval.pol` is a policy (i.e., a function of vectors `z` (previously observed outcomes) and `a` (previously observed assignments) of conforming length, allowing for `c()`), the horizon `n`, `delta` (a real number in $(0, 1/2)$, values closer to $1/2$ make the problem more difficult in the sense that the identity of the best arm is more difficult to determine), the number of replications `n.rep` and a parameter that regulates the setup (normal or uniform).

```
set.seed(1)
eval.pol <- function(pol, n, delta, n.rep, normal = TRUE){

  regret.matrix <- matrix(NA, nrow = n.rep, ncol = n)

  for(i in 1:n.rep){
    Z <- c()
    A <- c()

      for(j in 1:n){
        if(normal){
          Y_t <- rnorm(2, mean = c(1/2, delta))
        } else {
          Y_t <- c(runif(1, min = 0, max = 1/2),
                   runif(1, min = 0, max = delta))
        }
      as.t <- pol(Z, A)
      regret.matrix[i,j] <- Y_t[1] - Y_t[as.t]
      Z <- c(Z, Y_t[as.t])
      A <- c(A, as.t)
      }
    regret.matrix[i,] <- cumsum(regret.matrix[i,])
```

```
  }
return(regret.matrix)
}
```

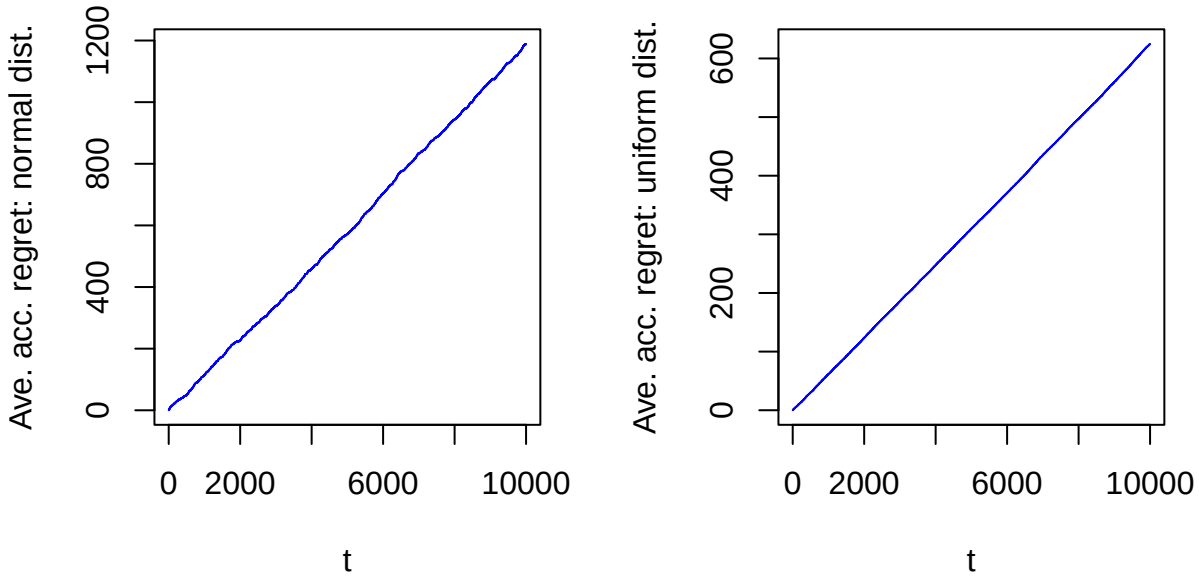
We can easily implement the uniform allocation policy:

```
unif.alloc <- function(Z, A){
  ((length(Z) + 1) %%  2)+1
}
```

Let's evaluate this policy in the two frameworks (normal and uniform) with $n = 10000$, $\delta = 1/4$ and 10 replications (over which we average to get an impression of the average regret):

```
R1_unif <- eval.pol(unif.alloc, 10000, 1/4, 10)
R2_unif <- eval.pol(unif.alloc, 10000, 1/4, 10, normal = FALSE)


par(mfrow = c(1, 2))
plot(apply(R1_unif, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: normal dist.")
plot(apply(R2_unif, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: uniform dist.")
```



Clearly, from the figures, we see that the policy does not "learn" (we know this from the definition of the policy, obviously). The regret grows linearly, that is we keep making the same sorts of mistakes *ad infinitum*. This is quite bad.

Slight modifications (which suffer from similar issues) are so-called semi-uniform strategies, for example the ϵ-greedy policy, which pulls the arm with the highest current average return in a proportion of $1 - \epsilon$ of the decisions, and there the remaining ϵ proportion is assigned randomly among the other arms. An implementation of that policy is given in the following:

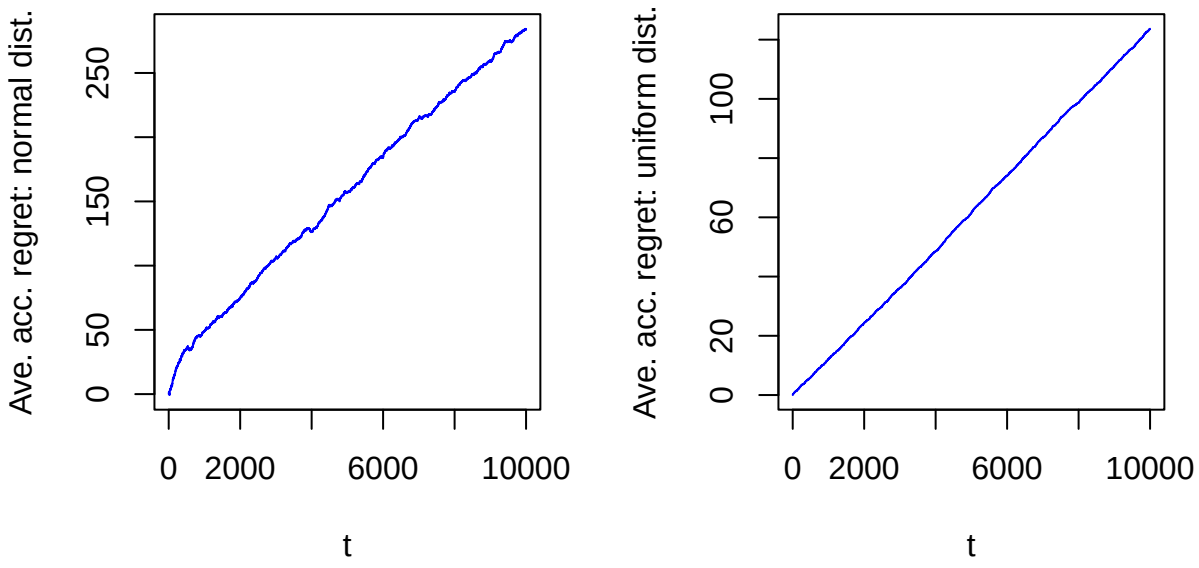
```
eps.greedy <- function(Z, A, eps = 0.1){
 t <- length(Z) + 1
 if(t <= 2){
    return(t)
 } else {
    randv <- rbinom(1, 1, 1-eps)
    h <- order(c(mean(Z[A == 1]), mean(Z[A == 2])))[2:1]
    if(randv == 1){
      return(h[1])
    } else {
```

```
      return(h[2])
    }
   }
}
```

We can evaluate the policy in the above-considered scenarios:

```
R1_eg <- eval.pol(eps.greedy, 10000, 1/4, 10)
R2_eg <- eval.pol(eps.greedy, 10000, 1/4, 10, normal = FALSE)

par(mfrow = c(1, 2))
plot(apply(R1_eg, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: normal dist.")
plot(apply(R2_eg, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: uniform dist.")
```



While it is clear that the regret behavior of the policy is significantly improved compared to the naive uniform assignment policy (which completely ignores the data gathered so far), the regret still increases linearly in n, which is also not so difficult to see theoretically: With probability ϵ, one chooses an arm with an average return that is smaller than the best one, so that the corresponding Δ_i is positive. Hence, $\epsilon\Delta_i$ is added to the

regret at each time point, leading to a linear increase in expected regret.

An immediate idea would now be to decrease ε as a function of t, which is called an "epsilon-decreasing" strategy. Furthermore, one could also start with an exploration phase of a given length in which one pulls all arms cyclically (or with the same probability) and then has a phase, where one exploits that knowledge. But this does not really lead to optimal procedures.

Before we continue with results on what is actually achievable with cleverly designed policies and which policies achieve these "lower bounds", it is up to you to think about how a policy better than uniform allocation could look like.

Now it's up to you!

What are your ideas, implement them and apply the function 'eval.pol' to your policy. Check how your policy performs in terms of n and for different values of δ. Note that your policy must not use that the second arm is inferior. Such a policy would then not work well if the scenario is reversed.

3 Regret lower bounds

The formal analysis of policies for multi-armed bandit problems would be easier if we knew what we can achieve in the best case. If the best-case performance is an average expected regret that grows linearly in time, then one cannot really achieve a performance improvement by considering policies other than the uniform allocation policy.

To this end, the literature mainly considers lower bounds on the accumulated expected regret (cf. in paticular Chapter 13 in Lattimore and Szepesvári [2020] for basic intuition). That is, one shows theoretically, that even the most ingeniously designed policy cannot be better than some **rate** of increase in the regret $R(\boldsymbol{\pi}, n)$ in n. Recall that once we fix the policy and the underlying distributions of the arms, the regret is a function of n. Hence, the question is how **quickly** does this function increase in n. The uniform allocation policy leads to a linear growth of that function in n. The question hence is if rates such as $\log(n)$ would be achievable (which grows much slower than linear, i.e., is sublinear).

What one can show are results of the following type: There exists a universal constant $c > 0$ such that for every policy $\boldsymbol{\pi}$ it holds that

$$\max_{\text{Set of outcome distributions}} R(\boldsymbol{\pi}, n) \geq c\sqrt{nK}.$$

Note that we consider here the worst-case performance of the policy over a given set of outcome distributions. This is relevant, because the outcome distributions are unknown, and one cannot incorporate knowledge of

them into the policy. Otherwise, it could be that a policy works well for certain distributions, but not for others. Hence, what one can do is to look at the worst-case behavior of a policy (with outcome distributions being among the distributions in a given set).

A policy $\boldsymbol{\pi}$ that attains the lower bound, i.e., for which $\max_{\text{Set of outcome distributions}} R(\boldsymbol{\pi}, n) \leq c'\sqrt{nK}$ for some $c' > 0$ is called minimax expected regret (rate) optimal. In terms of worst-case performance, no policy can achieve a better growth rate of the accumulated expected regret. In this sense, such a policy performs optimally. Obviously, this does not allow us to conclude much about the "non-worst-case" performance of such a policy.

The set of outcome distributions over which such a lower bound holds needs to be sufficiently "rich", e.g., needs to contain normal distributions with means in a non-empty interval $[\mu_{low}, \mu_{up}]$ or Bernoulli distributions with success probabilities in a non-empty interval $[p_{low}, p_{up}]$. Note that those two classes of distributions are actually quite narrow (i.e., are parametric). This means that even if one tries to take advantage about knowledge that the outcome distributions are normal (or Bernoulli), one cannot achieve a better performance of the policy's performance in terms of (worst case) maximal expected regret. In this sense, the results (lower bounds) are quite strong.

We also note that the rate $n \mapsto \sqrt{nK}$ is much lower than the linear rate $n \mapsto n$! Hence, there is considerable room for improvement.

4 Examples of policies that achieve the lower bounds

4.1 UCB policy

The main idea here is to gradually construct one-sided confidence intervals for the unknown means $\mu_1, \ldots, \mu_K$ (i.e., confidence intervals of the form $(-\infty, u_{i,t}]$, where each $u_{i,t}$ is a function of Z_t). One then tries to pull that arm that corresponds to the maximal upper bound of the confidence intervals.

A challenge one has to overcome is: the confidence intervals have to have coverage guarantees (uniformly over time), but should not be too conservative (since we really want to exploit and do not want to keep exploring unnecessarily often). The analysis is more involved than the construction of standard confidence intervals you are familiar with from introductory statistics classes. First of all, there is no normality assumption that would help (and to analyse the algorithms, we are need some finite sample, i.e., non-asymptotic guarantees). Secondly, the problem is sequential and is not based on a fixed sample. Furthermore, the upper bounds have to ensure that all arms are sufficiently well explored, i.e., that one does not ignore an arm which actually

corresponds to the best outcome.

Results that help to construct suitable confidence intervals are concentration inequalities on the sample average, i.e., upper bounds that decrease exponentially in sample size for probabilities of the form $\mathbb{P}(|\bar{x}_n - \mu| \geq \varepsilon)$, e.g., Hoeffding's inequality. We won't go into detail here.

An optimal choice of the upper bounds of the confidence intervals is the following one (cf. Chapter 8 in Lattimore and Szepesvári [2020], where a somewhat different notation is used)

$$u_{i,t} = \hat{\mu}_{i,t} + \sqrt{\frac{2\log(1 + t\log^2(t))}{S_i(t)}},$$

where $\hat{\mu}_{i,t}$ denotes the average of all observations among the previously observed outcomes that were assigned to arm i, and the value $S_i(t)$ denotes the number of such observations. Note that $u_{i,t}$ can be large for two reasons: (i) because the average is large, or (ii) because the second term in the definition of $u_{i,t}$ is large. The latter case arises if that arm (in comparison to the other arms) has not been sufficiently explored, in the sense that $S_i(t)$ is relatively small.

UCB policy

1. Assign each arm once.
2. In round $t = K+1, K+2, \ldots$ pull an arm in $\arg\max_{i=1,\ldots,K} u_{i,t}$.

Let's implement this policy and test is in the two frameworks we consider.

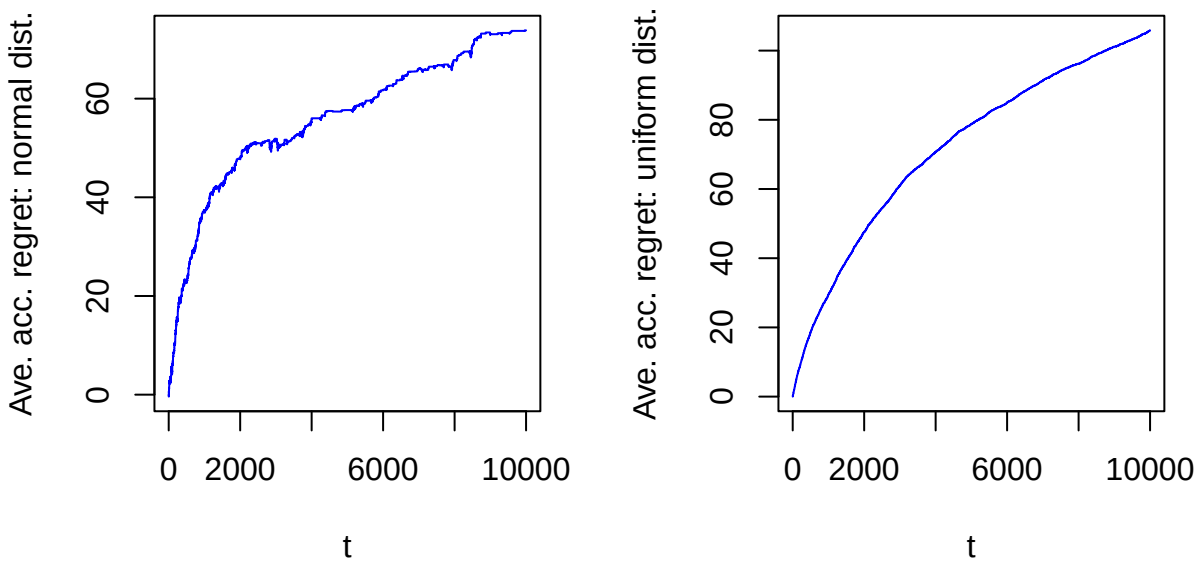
```
ucb <- function(Z, A){
  t <- length(Z) + 1
  if(t <= 2){
    return(t)
  } else {
    mu1 <- mean(Z[A == 1])
    S1 <- sum(A == 1)
    u1 <- mu1 + sqrt(2*log(1+t*log(t)^2)/S1)
    mu2 <- mean(Z[A == 2])
    S2 <- sum(A == 2)
    u2 <- mu2 + sqrt(2*log(1+t*log(t)^2)/S2)
    return(which.max(c(u1, u2))[1])
  }
```

```
}
```

Finally, let's evaluate this policy in the two frameworks:

```
R1_ucb <- eval.pol(ucb, 10000, 1/4, 10)
R2_ucb <- eval.pol(ucb, 10000, 1/4, 10, normal = FALSE)


par(mfrow = c(1, 2))
plot(apply(R1_ucb, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: normal dist.")
plot(apply(R2_ucb, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: uniform dist.")
```



On a theoretical level, one can show that the maximal expected regret of the UCB policy is bounded from above by a multiple of $\sqrt{Kn\log(K)}$. Hence, there is still a logarithmic factor in K present in the upper bound, which is not present for the lower bounds we have seen in the previous section.

4.2 MOSS policy

The MOSS policy was invented to precisely remove the $\sqrt{\log(K)}$ factor in the upper bound of the UCB policy. The MOSS policy itself (more specifically its anytime version due to Degenne and Perchet [2016]) is

similar to the UCB policy in that it also relies on one-sided confidence sets. Here, the only difference is that the upper bound is chosen as

$$v_{i,t} = \hat{\mu}_{i,t} + \sqrt{\frac{\max\left(0, \log(t/(KS_i(t)))\right)}{S_i(t)}}.$$

That one can remove the logarithmic factor in K in the upper bound was a breakthrough in the asymptotic analysis of multi-armed bandits.

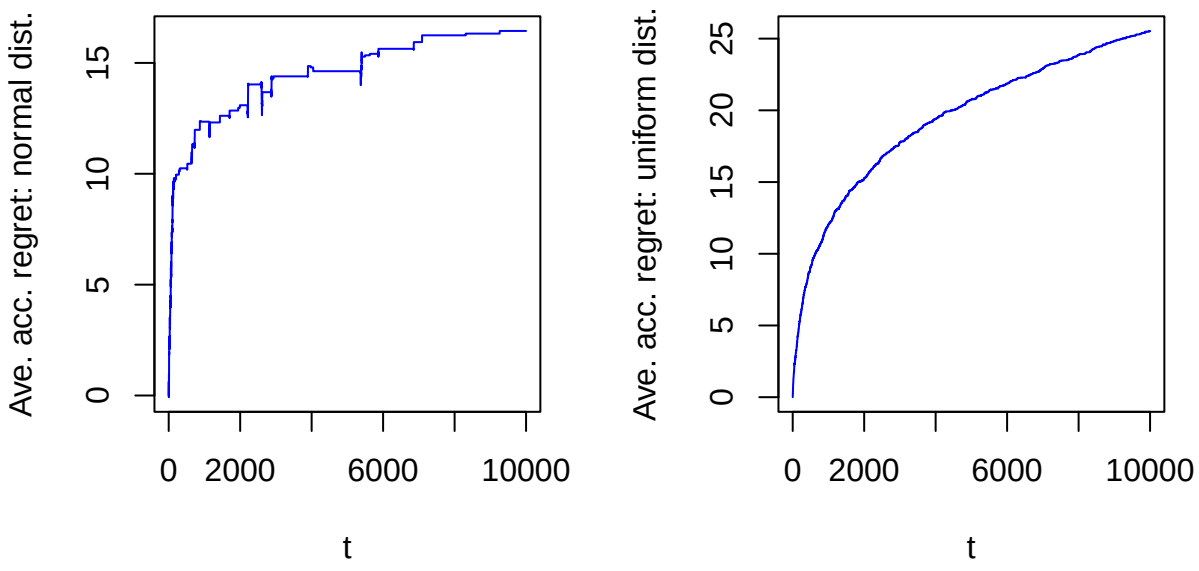
Since this is only a slight modification of the UCB policy, implementing MOSS is easy:

```
moss <- function(Z, A){
  t <- length(Z) + 1
  if(t <= 2){
    return(t)
  } else {
    mu1 <- mean(Z[A == 1])
    S1 <- sum(A == 1)
    u1 <- mu1 + sqrt((max(c(0, log(t/(2*S1))))))/S1)
    mu2 <- mean(Z[A == 2])
    S2 <- sum(A == 2)
    u2 <- mu2 + sqrt((max(c(0, log(t/(2*S2))))))/S2)
    return(which.max(c(u1, u2)))
  }
}
```

Finally, let's evaluate this policy in the two frameworks:

```
R1_moss <- eval.pol(moss, 10000, 1/4, 10)
R2_moss <- eval.pol(moss, 10000, 1/4, 10, normal = FALSE)


par(mfrow = c(1, 2))
plot(apply(R1_moss, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: normal dist.")
plot(apply(R2_moss, 2, mean), type = "l", col = "blue",
     xlab = "t", ylab = "Ave. acc. regret: uniform dist.")
```



In particular, when comparing the performance of that policy to the performance of uniform allocation, or an ϵ version thereof, we see that the development of cleverly designed policies can dramatically reduce the regret (or in other words, can dramatically increase the reward gained).

References

Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

Arthur Charpentier, Romuald Elie, and Carl Remlinger. Reinforcement learning in economics and finance. *Computational Economics*, pages 1–38, 2021.

Rémy Degenne and Vianney Perchet. Anytime optimal algorithms in stochastic multi-armed bandits. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1587–1595, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/degenne16.html.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.