

COMPSCI 4SD3

Data-Driven Algorithms for Sequential Decision Making

Assignment 1

Winter 2026

Sivan Sabato, McMaster University

Due Date: See Avenue to Learn

- Submit on the course’s Avenue to Learn website.
- The answers to all the questions (including questions about the code part) must be typeset in Latex and submitted as a single pdf file (call it **assignment.pdf**). Submit the pdf file into the “questions part” assignment component in A2L.
- The code for the programming task (Question 3 below) must be submitted as a single Python file (call it **assignment1.py**) into the “code part” assignment component in A2L.
- This assignment is worth 10% of the final course grade.
- For questions about the assignment, use the “Q and A” channel on the course Teams.
- This is an individual assignment. Each student should solve and submit it on their own.
- Use of generative AI for assignments is **strictly forbidden** in this course, and will be considered a violation of academic integrity.

1. **[15 points]** Answer the following questions using Hoeffding’s inequality and our adaptation to a general reward range.
 - (a) Arm a has rewards in the range $[0, 10]$. We pull arm a 32 times. We use Hoeffding’s inequality to guarantee that the probability that $|\hat{\mu}_a(n) - \mu_a| < \epsilon$ is at least 90%. What is the smallest value of ϵ that would make the statement above hold?
 - (b) Arm b has rewards in the range $[0, 1]$. The expected reward of arm b is $\mu_b = 0.5$. We want to pull the arm m times, such that with a probability of at least 97%, $\hat{\mu}_b(n) \leq 0.7$. What is the smallest m that we can use?
 - (c) There are three arms a_1, a_2, a_3 . The arms have reward ranges $[0, 1]$, $[0, 2]$, and $[0, 4]$ respectively. We want to pull each arm m times, such that the following event holds with probability at least 93%:

$$\forall i \in \{1, 2, 3\}, |\hat{\mu}_{a_i}(n) - \mu_{a_i}| < 0.1.$$

What is the smallest m we can use?

2. **[28 points]** In the algorithm “successive elimination with known gaps”, consider the following scenario: $\alpha = 4, \delta_0 = 0.05, \mu_{a_1} = 3, \mu_{a_2} = 2, \mu_{a_3} = 1.9, \mu_{a_4} = 1.8$. In the questions below, assume that the input to the algorithm specifies the true gaps $\Delta_2, \dots, \Delta_4$. In addition, assume that in each round i ,

$$|\hat{\mu}_{a_1}^{(i)} - \mu_{a_1}| < \epsilon_i \quad \text{and} \quad |\hat{\mu}_{j_i}^{(i)} - \mu_{j_i}| < \epsilon_i,$$

where j_i is the arm with the smallest expected reward that is still in S in round i .

- (a) How many arm pulls will occur in round 1 for each arm in S ?
 - (b) Prove that a_1 is not eliminated in round 1.
 - (c) Will arm 4 necessarily be eliminated in round 1? Explain why or why not.
 - (d) How many arm pulls will occur in round 2 for each arm in S ?
 - (e) Prove that a_1 is not eliminated in round 2.
 - (f) In Round 3, which arms (if any) are definitely in S ? Explain.
 - (g) Which arms (if any) are definitely not in S ? Explain.
3. [7 points] In successive elimination with unknown gaps, prove that if the “bad events” that we defined in class do not occur and

$$t > \frac{8 \cdot \alpha^2 \ln(2/\delta_t)}{\Delta_i^2},$$

then arm i is eliminated by round t .

4. [30 points] Programming task; You do not need to include anything as answer to this question in your submitted pdf file. Only submit the Python file to the “code part” of the assignment (see submission instructions in the box above).

Implement two algorithms that we learned in class:

- (a) The uniform sampling algorithm
- (b) The successive elimination algorithm with known gaps.

Technical instructions:

- Download the necessary files from the coding assignment component on A2L.
 - Read the `README.pdf` file. Follow the instructions there to set up a python environment with the correct package versions, so that your code runs correctly when we test it on our system.
 - Each of the algorithms should be implemented as a function. The provided file `assignment1.py` lists the required function declarations. You need to fill in the missing code.
 - In addition to the algorithm input parameters that we learned in class, the functions you will implement include an additional integer parameter called `pull_reduction`. Its default value should be 1. When this parameter is 1, the algorithm runs just as we learned in class. When this parameter is larger than 1, you should divide all of the numbers of pulls that the algorithm calculates by this value. For instance, if the algorithm calculated that you need to pull an arm 36 times, and `pull_reduction` is 3, then you should pull the arm only $36/3 = 12$ times. If the result is not an integer, you should use the ceiling of the result.
 - Before submitting your code, make sure `usage_example.py` runs correctly when combined with your submitted file `assignment1.py` and the provided file `multi_armed_bandit.py`, which implements the arm pulls. This is a basic check to verify that you have followed the required definitions.
 - The run of `usage_example.py` with your implementation should take well under a second.
 - Do not count on `usage_example.py` as the only test. Test your code in difference scenarios to make sure it works correctly.
 - You can use the functions `_get_true_mean` for each arm in your tests to verify that your algorithm works well, but you are not allowed to use them in your submitted code.
5. [20 points] Run each of the two algorithms that you implemented in the previous question on the following set of arms:

1. Bernoulli, $p = 0.42$

2. Bernoulli, $p = 0.43$
3. Normal distribution with $\mu = 15.2$ and $\sigma = 2.5$.
4. Normal distribution with $\mu = 16$ and $\sigma = 1$.
5. Uniform on $[0, 2.5]$
6. Uniform on $[1.24, 1.26]$

- (a) What input arguments (other than the list of arms) do you need to provide each algorithm to find the best arm with a probability at least 0.8? Explain the calculation of each value.
- (b) Run each algorithm with the above input arguments. What was the number of pulls for each algorithm? Which one was smaller and why?
- (c) Run each algorithm with the above input arguments 100 times. For each algorithm, what was the percent of successful runs, in which the algorithms provided the best arm? Was it more or less than the value you expected? Explain what could be the reason for the difference.
- (d) Run the algorithms 100 times again, this time setting `pull_reduction = 34`. For each algorithm, what was the percent of successful runs, where the algorithms provided the best arm? What can you conclude about the formulas that the algorithms are using for the number of required arm pulls?
- (e) **[Bonus question: 5 additional points]** When you ran the successive elimination algorithm 100 times with `pull_reduction = 1`, when the algorithm was successful, did it always remove the arms in the order from the worst to the best? Describe what you observed and explain what you think was the reason for this.