EMORY UNIVERSITY
Department of Computer Science
CS 334 Section 2 — Machine Learning
Spring 2025

Homework 6, Issued: Sun. 4/6, Due: Sun. 4/20 at 11:59pm

Submission Instructions: The homework is due on Gradescope in **two** parts.

- **Upload PDF to HW6-Written**: Create a single high-quality PDF with your answers to the non-coding problems. Your submission may be typed or handwritten, and pages must be tagged with relevant parts on Gradescope.
- **Submit your code to HW6-Code**: Questions marked with 🛠 on the left margin are graded by a Python autograder. Your submission must include the following file(s): `bandit.py`, `README.txt`. You may submit multiple times; only the last submission will be considered. Please include the following ***SIGNED*** *honor code statement*:

```
THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING CODE
WRITTEN BY OTHER STUDENTS OR LARGE LANGUAGE MODELS SUCH AS CHATGPT.
/* Your_Name_Here */
I collaborated with the following classmates for this homework:
<names of classmates>
```

1 Multi-Armed Bandit [30 pts]

In this question, we will investigate how the hyperparameters of the three bandit algorithms we covered in lecture affect their performance. You'll first implement the algorithms and then run them on a multi-armed bandit problem. The provided skeleton code in `bandit.py` should only require you to fill in a few blanks. **A correct implementation requires < 20 lines of code and take < 3 minutes to run.**

(a) Implement the following three functions by filling in the missing lines in the skeleton code.

🛠 i. [5 pts] Implement `epsilon_greedy(mab, T, epsilon)`. Note we already implemented ϵ-greedy action selection for you; feel free to adapt these lines for subsequent parts.

🛠 ii. [5 pts] Implement `optimistic_initialization(mab, T, Q1)` with greedy action selection.

🛠 iii. [5 pts] Implement `upper_confidence_bound(mab, T, c)` with greedy action selection using the UCB scores.

All functions share the same interface: they run algorithm for T steps on the multi-armed bandit problem `mab` with an exploration probability of `epsilon`. The function returns three metrics that track the performance at each time step: reward at step t, whether the action is optimal at step t, and the cumulative regret up to step t defined as $L_t = \sum_{i=1}^{t}(\mu^* - \mu_{a_t})$ where $\mu^* = \max_a \mu_a$.

Submit your code to the autograder and debug your implementation before moving to the next part.

continued on the next page...

(b) Let's now apply these functions and run some simulation.

We'll consider a 5-armed bandit problem, where each arm's reward follows a Bernoulli distribution with means $(\mu_1, \mu_2, \mu_3, \mu_4, \mu_5) = (0.1, 0.275, 0.45, 0.625, 0.8)$. Recall that for a Bernoulli random variable $X \sim \text{Bernoulli}(p)$, then $X = 1$ with probability p, we have $X = 0$ with probability $1 - p$, and $\mathbb{E}[X] = p$. The order of the 5 arms is shuffled and your algorithms should not hard-code the optimal arm index when selecting actions.

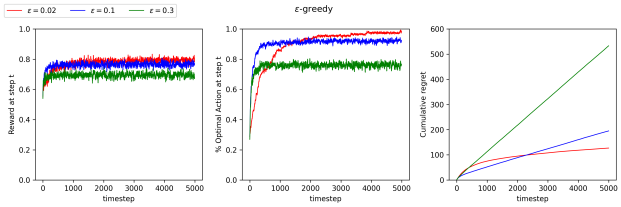
Study the provided code in the main function and run the script. The script performs three sets of experiments, one for each algorithm, and generate some plots that track the learning progress of each algorithm. For clearer visualization of the trends, results are averaged over 100 runs and the learning curves are smoothed. The goal here is to test the effect of their hyperparameters.

i. ϵ-greedy with $\epsilon \in \{0.02, 0.1, 0.3\}$.

[1 pts] Attach the generated plot named `bandit__epsilon_greedy.png`. Your plot should look similar to the following.

[2 pts] Comment on the how the performance varies for larger vs smaller ϵ values. Specifically, compare the the initial speed of learning, the speed of convergence, and the asymptotic performance after convergence.

[1 pt] Explain the differences in relation to how ϵ affects the exploration-exploitation trade-off.



ii. Optimistic initialization with $Q_0 \in \{0, 5, 50\}$.

[1 pts] Attach the generated plot named `bandit__optimistic_initialization.png`.

[2 pts] Comment on the how the performance varies for different Q_0 values.

[1 pt] Explain the differences in relation to how Q_1 affects the exploration-exploitation trade-off.

iii. Upper confidence bounds with $c \in \{0.2, 1, 2\}$

[1 pts] Attach the generated plot named `bandit__upper_confidence_bounds.png`.

[2 pts] Comment on the how the performance varies for different c values.

[1 pt] Explain the differences in relation to how c affects the exploration-exploitation trade-off.

(c) i. [2 pts] Compare the performance across the three algorithms. Which algorithm tends to be the best, and which tends to be the worst? What algorithm and hyperparameter would you use and why?

ii. [1 pt] Only one of the three algorithms has "sublinear regret", i.e., cumulative regret grows slower than $O(T)$. Based on your simulation results, which algorithm has sublinear regret?

2 Collaborative Filtering [16 pts]

At the end of the semester, students provide feedback to the courses they took via an online evaluation form. Wouldn't it be nice if the evaluation form is pre-filled for you! In this question, we'll try to predict the course ratings that haven't been filled out by students.

Let's assume that students rate courses on a scale of 1 to 5. We have a dataset of 5 students and 3 courses, represented by matrix Y.

$$Y = \begin{bmatrix} 5 & 3 & ? \\ ? & 2 & 3 \\ ? & 1 & ? \\ 5 & ? & ? \\ ? & 5 & 2 \end{bmatrix}$$

Here the rows correspond to students, and the columns correspond to courses. Each cell Y_{ai} corresponds to the rating student a assigned to course i. Notice that not all the courses have been rated. We will try to estimate these missing values via a low-rank approximation to Y denoted as $\hat{Y} = UV^T$, where U is $5 \times d$ and V is $3 \times d$. We will use the matrix factorization approach and minimize the following cost function:

$$J(U,V) = \frac{1}{2} \sum_{(a,i) \in D} (Y_{ai} - [UV^T]_{ai})^2 + \frac{\lambda}{2} \sum_{a=1}^{5} \|\vec{u}^{(a)}\|^2 + \frac{\lambda}{2} \sum_{i=1}^{3} \|\vec{v}^{(i)}\|^2$$

Here, $[UV^T]_{ai} = \vec{u}^{(a)} \cdot \vec{v}^{(i)}$. D is the set of observed entries. Suppose we are trying to find a rank-1 approximation to Y, in other words $d = 1$. Let's assume $\lambda = 1$. After some number of iterations, we obtain $U = [6, 2, 3, 3, 4]^T$ and $V = [4, 2, 1]^T$.

(a) [3 pts] What is the squared error associated with the entry $\hat{Y}_{53}$?

(b) Assume we now keep the v's fixed and solve for the new u's.

i. [6 pts] Write down the optimization problem we must solve in terms of Y, $u^{(5)}$, $v^{(i)}$, and λ to obtain the updated feature vector associated with the 5th student. You can use D_a to denote the set of observed entries for student a.

$$\min_{\_\_} \frac{1}{2} \sum_{\_\_} (\_\_\_\_\_)^2 + \frac{\lambda}{2} \_\_\_\_\_$$

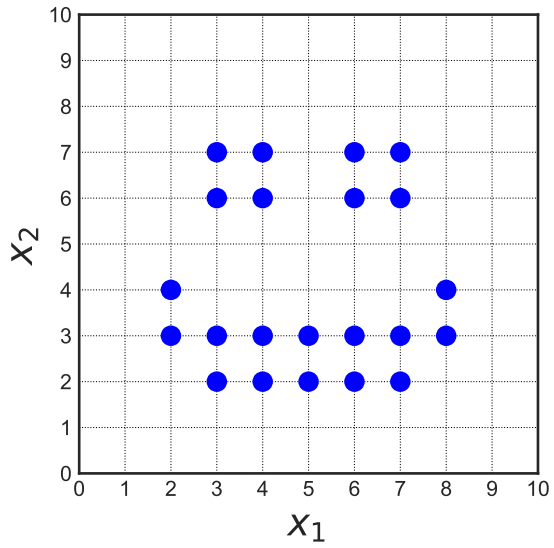
ii. [4 pts] Solve this optimization problem. What is the new feature vector for the 5th student?

(c) [3 pts] Using the updated $u^{(5)}$, what is the new prediction for $\hat{Y}_{53}$?

3 Clustering [14 pts]

(a) [3 pts] For clustering using k-means, why should one be consistent in how ties are broken?

Consider the data plotted in figure below, which consists of 22 points that make up a smiley face.



(b) Suppose we initialized three centroids at coordinates $\vec{z}^{(1)} = [2, 2]^T$, $\vec{z}^{(2)} = [8, 2]^T$, and $\vec{z}^{(3)} = [5, 7]^T$. We then run k-means clustering with $k = 3$ and Euclidean distance as the distance measure. Assume that ties are broken by the following order according to which the algorithm will favor assigning points to clusters: $C_1 > C_2 > C_3$.

i. [2 pts] How many iterations does it take for k-means to converge? We count the number of iterations as the number of times that centroids are updated.

ii. [4 pts] Indicate the final clusters obtained (after convergence) and the corresponding centroids.

(c) [5 pts] Suggest an initialization (the initial centroids for 3 clusters) with which the algorithm would converge to a different solution.