CS 234 Winter 2022
HW 5
Due: Mar 11 at 11:59 pm (PST)

For submission instructions please refer to the website. For all problems, if you use an existing result from either the literature or a textbook to solve the exercise, you need to cite the source.

1 Offline Policy Evaluation [15 pts]

Motivation Previously, you have learned OPE techniques such as importance sampling and fitted Q evaluation (FQE). However, we can estimate value of a new policy in a much simpler form. The value of a new policy π can be estimated as:

$$\rho(\pi) = \mathbb{E}_{(s,a)\sim d^{\mathcal{D}}, r\sim R(s,a)}\left[w_{\pi/\mathcal{D}}(s,a)\cdot r\right], \quad \omega_{\pi/\mathcal{D}}(s,a) = \frac{d^\pi(s,a)}{d^{\mathcal{D}}(s,a)}$$

This avoids the importance sampling calculation which has a high variance when the horizon H is large, and function approximation error in FQE (if we use function approximation to learn the Q function). However, estimating the ratio $\omega_{\pi/\mathcal{D}}(s,a)$ is not easy because it contains state visit distributions: even if we can estimate $d^{\mathcal{D}}(s,a)$, we still have trouble estimating $d^\pi(s,a)$ without access to the MDP.

In the following problem, you'll view OPE as an optimization problem, where the solution to the optimization objective will be $\omega_{\pi/\mathcal{D}}(s,a)$.

Problem Consider an infinite-horizon, discounted MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma\rangle$ where $\gamma \in [0,1)$ and the state-action space is finite ($|\mathcal{S}\times\mathcal{A}| < \infty$). For any policy $\pi : S \to \Delta(A)$, recall that the discounted stationary state distribution is defined for any state $s \in S$ as

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \beta_t^\pi\left(s_t = s\right)$$

where $\beta_t^\pi\left(s_t = s\right)$ denotes the probability that the (random) state s_t encountered by policy π at timestep t is equal to s:

$$\beta_t^\pi(s) := \Pr\left(s = s_t \mid s_0 \sim \beta, a_k \sim \pi\left(s_k\right), s_{k+1} \sim T\left(s_k, a_k\right) \text{ for } 0 \le k < t\right)$$

Let $\beta \in \Delta(S)$ be an initial state distribution such that $\beta_0^\pi = \beta(s)$ for all policies π and any state $s \in S$. Recall that we came across discounted stationary state distributions in Assignment 2 while proving the Performance Difference lemma.

Let's consider the batch reinforcement-learning setting where we have a dataset D of transitions (s, a, r, s') sampled in a completely unknown fashion. While $\mathcal{D}$ is not a policy, we will denote the distribution over state-action pairs sampled from $\mathcal{D}$ as $d^{\mathcal{D}}(s, a)$. For the remaining questions, define

$$d^\pi(s, a) = d^\pi(s)\pi(a \mid s)$$

(a). - **5 pts.** For any two arbitrary sets $\mathcal{X}$ and $\mathcal{Y}$, we denote the class of all bounded functions mapping from $\mathcal{X}$ to $\mathcal{Y}$ as $\{\mathcal{X} \to \mathcal{Y}\}$. Let $x \in \{S \times A \to \mathbb{R}\}$ such that for all $(s, a) \in S \times A$,

$$J(x) = \frac{1}{2}\mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}\left[x(s, a)^2\right] - \mathbb{E}_{(s,a)\sim d^\pi}[x(s, a)].$$

Let $x^\star$ be the minimizer of $J(x)$. Prove that $x^\star(s, a) = \omega_{\pi/\mathcal{D}}$ for any state-action pair $(s, a) \in S \times A$.

Hint: $J(x)$ is of the form $\frac{1}{2}nx^2 - mx$ and we want to find the x that minimizes $J(x)$.

Let

$$J(x) = \frac{1}{2}\mathbb{E}_{(s,a)\sim d^{\mathcal{D}}}\left[x(s, a)^2\right] - \mathbb{E}_{(s,a)\sim d^\pi}[x(s, a)]$$

We get

$$\nabla_x J(x) = d^{\mathcal{D}}(s, a)x(s, a) - d^\pi$$

Set $\nabla_x J(x) = 0$, we get

$$x(s, a) = \frac{d^\pi(s, a)}{d^{\mathcal{D}}(s, a)} = \omega_{\pi/\mathcal{D}}(s, a)$$

One might wonder why we can't just stop right here after coming up with a viable density-ratio estimator. Notice, however, that approximating the second expectation would require running rollouts of policy π in the environment, which directly contradicts the OPE setting of this problem. So, we must work a little bit harder to come up with an objective that averts this dependence.

(b). - **5 pts** If we can solve the optimization problem from part (a), then we can find $\omega_{\pi/D}$. But the objective function in the previous part is problematic since the second term requires samples from the stationary state-action distribution $d^\pi(s, a)$. Consider a value function $\nu \in \{S \times A \to \mathbb{R}\}$ whose rewards are given by some $x \in \{S \times A \to \mathbb{R}\}$ such that for all $(s, a) \in S \times A$,

$$\nu(s, a) := x(s, a) + \gamma\mathbb{E}_{s'\sim\mathcal{T}(\cdot|s,a)}\left[\mathbb{E}_{a'\sim\pi(\cdot|s')}\left[\nu\left(s', a'\right)\right]\right]$$

Prove that $\mathbb{E}_{(s,a)\sim d^\pi}[x(s, a)] = (1 - \gamma)\mathbb{E}_{s_0\sim\beta}\left[\mathbb{E}_{a_0\sim\pi(\cdot|s_0)}\left[\nu\left(s_0, a_0\right)\right]\right]$.

Hint: Use β_t, the state visitation probability at step t when following π.

$$\begin{aligned}
\mathbb{E}_{(s,a)\sim d^\pi}[x(s, a)] &= \mathbb{E}_{(s,a)\sim d^\pi}\left[\nu(s, a) - \gamma\mathbb{E}_{s'\sim T(s,a),a'\sim\pi(s')}\left[\nu\left(s', a'\right)\right]\right] \\
&= (1 - \gamma)\sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{s\sim\beta_t,a\sim\pi(s)}\left[\nu(s, a) - \gamma\mathbb{E}_{s'\sim T(s,a),a'\sim\pi(s')}\left[\nu\left(s', a'\right)\right]\right] \\
&= (1 - \gamma)\sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{s\sim\beta_t,a\sim\pi(s)}[\nu(s, a)] - (1 - \gamma)\sum_{t=0}^{\infty}\gamma^{t+1}\mathbb{E}_{s\sim\beta_{t+1},a\sim\pi(s)}[\nu(s, a)] \\
&= (1 - \gamma)\mathbb{E}_{s_0\sim\beta}\left[\mathbb{E}_{a_0\sim\pi(\cdot|s_0)}\left[\nu\left(s_0, a_0\right)\right]\right]
\end{aligned}$$

Step 2 to step 3 happens when we pull out γ on the second term, then it all cancels out except for γ^0 (the 0-th term). And we end up with s_0 and a_0.

It's worth noting that this problem could also be solved without the hint by leveraging one of the results proven in Assignment 2. Namely, that for any arbitrary function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we have the following identity:

$$\mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)\right] = \frac{1}{(1-\gamma)} \mathbb{E}_{s \sim d^\pi}\left[\mathbb{E}_{a \sim \pi(\cdot|s)}\left[f(s,a)\right]\right].$$

The Bellman equation provided above implies that we can also express $\nu(s,a)$ as a discounted sum of rewards given by x:

$$\nu(s,a) = \mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t x(s_t, a_t) \mid s_0 = s, a_0 = a\right].$$

If we divide through our claim by $(1-\gamma)$, we have

$$\begin{aligned}
\frac{1}{(1-\gamma)} \mathbb{E}_{(s,a) \sim d^\pi}[x(s,a)] &= \mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t x(s_t, a_t)\right] \\
&= \mathbb{E}_{s_0 \sim \beta}\left[\mathbb{E}_{a_0 \sim \pi(\cdot|s_0)}\left[\mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t x(s_t, a_t) \mid s_0, a_0\right]\right]\right] \\
&= \mathbb{E}_{s_0 \sim \beta}\left[\mathbb{E}_{a_0 \sim \pi(\cdot|s_0)}\left[\nu\left(s_0, a_0\right)\right]\right],
\end{aligned}$$

where the penultimate line employs the tower property of expectation. Observe that this solution, while equally correct and mathematically simpler, is a bit more conceptually complex than the previous solution advocated for by the hint.

(c). - **5 pts** The result from part (b) allows us to remove the expectation under $d^\pi(s,a)$ in $J(x)$. Rewrite $J(x)$ as $J(\nu)$ using the result from part (b). Your objective should no longer have the expectation under $d^\pi(s,a)$. You should simplify your answer by using the zero-reward Bellman operator:

$$\mathcal{B}^\pi \nu(s,a) = \gamma \mathbb{E}_{s' \sim T(s,a), a' \sim \pi(s')}\left[\nu\left(s', a'\right)\right].$$

Combined with your result from part (a), what is the minimizer of $J(\nu)$?

The objective is:

$$\min_{\nu : S \times A \to \mathbb{R}} J(\nu) := \frac{1}{2} \mathbb{E}_{(s,a) \sim d^{\mathcal{D}}}\left[\left(\nu - \mathcal{B}^\pi \nu\right)(s,a)^2\right] - (1-\gamma) \mathbb{E}_{s_0 \sim \beta, a_0 \sim \pi(s_0)}\left[\nu\left(s_0, a_0\right)\right]$$

Combine the result with 1(a), we know that the minimizer will be

$$\left(\nu^* - \mathcal{B}^\pi \nu^*\right)(s,a) = w_{\pi/\mathcal{D}}(s,a)$$

2 RL for personalized recommendations [60pts]

One of the most influential applications of RL in the real-world is in generating personalized recommendations for videos, movies, games, music, etc. Companies such as Netflix (Kawale and Chow (2018)), Spotify (Mehrotra (2018)), Yahoo (Li et al. (2010)) and Microsoft (Swaminathan et al. (2017)) use contextual bandits to recommend content that is more likely to catch the user's attention. Generating recommendations is an important task for these companies — the value of Netflix recommendations to the company itself is estimated at $1 billion (Gomez-Uribe and Hunt (2016)).

Content recommendations take place in a dynamical system containing feedback loops (Beer (1995)) affecting both users and producers. Reading recommended articles and watching recommended videos changes people's interests and preferences, and content providers change what they make as they learn what their audience values. However, when the system recommends content to the user, the user's choices are strongly influenced by the set of options offered. This creates a feedback loop in which training data reflects *both* user preferences and previous algorithmic recommendations.

In this problem, we will investigate how *video creators* learn from people's interactions with a recommendation system for videos, how they change the videos they produce accordingly, and what these provider-side feedback loops mean for the users. Dynamics similar to the ones we investigate here have been studied in newsrooms as journalists respond to real-time information about article popularity (Christin (2018)), and on YouTube as video creators use metrics such as clicks, minutes watched, likes and dislikes, comments, and more to determine what video topics and formats their audience prefers (Christin and Lewis (2021)).

We have created a (toy) simulation that allows you to model a video recommender system as a contextual bandit problem.[1] In our simulation, assume we have a certain fixed number of users N_u. Each user has a set of preferences, and their preference sets are all different from one another. We start off with some number of videos we can recommend to these users. These videos correspond to the arms of the contextual bandit. Initially there are N_a arms. Your goal is to develop a contextual bandit algorithm that can recommend the best videos to each user as quickly as possible.

In our Warfarin setting from assignment 4, N_a was fixed: we always chose from three different dosages for all patients. However, video hosting and recommendation sites like YouTube are more dynamic. Content creators monitor user behavior and add new videos (i.e. arms) to the platform in response to the popularity of their past videos. In other words, N_a keeps increasing over time.

How does this change the problem to be solved? Are we still in the bandit setting or is this now morphing into an RL problem? For now, we will treat it as a bandit problem. Remember that the number of users is static: N_u is a constant and doesn't change. In the coding portion of this assignment, you will study the effect of adding new arms into the contextual bandit setup, the different strategies we can employ to add these arms and measure how they affect performance.

Implementational details of the simulator

Most of the simulator has been written for you but the details might be useful in analysing your results. The only parts of the simulator you will need to write are the different strategies used to add more arms to the contextual bandit.

[1]According to Covington et al. (2016), YouTube does not currently use RL for their recommendations, but other video recommendation systems do, as noted above.

The simulator is initialized with N_u users and N_a arms where each user and arm is represented as a feature vector of dimension d. Each element of these vectors is initialized i.i.d from a normal distribution. When reset, the simulator returns a random user context vector from the set of N_u users. When the algorithm chooses an arm, the simulator returns a reward of 0 if the arm chosen was the best arm for that user and -1 otherwise.
We will be running the simulator for T steps where each step represents one user interaction. After every K steps, we add an arm to the simulator using one of three different strategies outlined below. Go through the code for the simulator in `sim.py`. Most of the simulator is implemented for you: the only method you will need to implement is `update_arms()`.

Implementing the Bandit algorithm [10pts]

For this assignment we will be using the Disjoint Linear Upper Confidence Bound (LinUCB) algorithm from Li et al. (2010). You have already implemented this in assignment 4 but you will have to make some minor changes to your code to account for the fact that the number of arms could keep increasing now. You will need to write about 15 lines of code in `hw5.py` for this part, most of which should come directly from assignment 4. The hints provided in `hw5.py` should help you with this.

Implementing the arm update strategies [30pts]

We will now implement three different strategies to add arms to our simulator. Each arm is associated with its true feature vector θ_a^*. This is the d-dimensional feature vector we assigned to each arm when we initialized the simulator. This is the θ the LinUCB algorithm is trying to learn for each arm through A and b. When we create new arms, we need to create these new feature vectors as well.
When coming up with strategies to add arms, we need to put ourselves in the shoes of content creators and think about how we want to optimize for the videos that go up on our channels. When making such decisions, we only consider the previous K steps since we added the last arm. Consider the three strategies outlined below:

1. **Popular:** For the last K steps, pick the two most popular arms and create a new arm with the mean of the true feature vectors of these two arms. For example, assume a_1 and a_2 were the two most chosen arms in the previous K steps with true feature vectors θ_1^* and θ_2^* respectively. Now create a new arm a with $\theta_a^* = \frac{\theta_1^*+\theta_2^*}{2}$.
   In the real world, this is similar to a naive approach where content creators create a new video based on their two most recommended videos from the last month.

2. **Corrective:** For the last K steps, consider all the users for whom we made incorrect recommendations. Assume we know what the best arm would have been for each of those users. Consider taking corrective action by creating a new arm that is the weighted mean of all these true best arms for these users. For example, say for the last K steps, we got $n_1 + n_2$ predictions wrong where the true best arm was a_1 n_1 times and a_2 n_2 times. Create a new arm a with $\theta_a^* = \frac{n_1\theta_1^*+n_2\theta_2^*}{n_1+n_2}$.
   In the real world, this is analogous to content creators adapting their content to give their viewers what they want to watch based on feedback from viewers about their preferences.

3. **Counterfactual:** Consider the following counterfactual: For the previous K steps, had there existed an additional arm a, what would its true feature vector θ_a^* have to be so that it would

have been better than the chosen arm at each of those K steps? There are several ways to pose this optimization problem. Consider the following formulation:

$$\theta_a^* = \arg\max_{\theta} \frac{1}{2} \sum_{i=1}^{K} (\theta^T x_i - \theta_i^T x_i)^2$$

Here x_i is the context vector of the user at step i. θ_i is the true feature vector of the arm chosen at step i. We can now optimize this objective using batch gradient ascent.

$$\theta_a \leftarrow \theta_a + \eta \frac{\partial L}{\partial \theta}$$

Here η is the learning rate and $L = \sum_{i=1}^{K} (\theta^T x_i - \theta_i^T x_i)^2$.
We can find $\frac{\partial L}{\partial \theta}$ directly as $\sum_{i=1}^{K} (\theta^T x_i - \theta_i^T x_i) x_i$. We can write the update rule as

$$\theta_a \leftarrow \theta_a + \eta \sum_{i=1}^{K} (\theta_a^T x_i - \theta_i^T x_i) x_i$$

In the real world, this is akin to asking the question, "What item could I have recommended in the past K steps that would have been better than all recommendations made in the past K steps?" In asking this, the creator aims to produce a new video that would appeal to all users more than the video that was recommended to them.

Implement these three methods in the `update_arms()` function in `sim.py`. This should be about 25 lines of code. Hints have been provided in the form of comments. **Each method is worth 10 points.**

Analysis [15pts]

For all the experiments in this section, we will initialize the simulator with 25 users and 10 arms each represented by a feature vector of dimension 10. We will run the simulation for a total of 10000 steps and use $\alpha = 1.0$ for the LinUCB algorithm. All these arguments have already been set for you and you will not have to change them.
For the questions below, please answer with just **2-3 sentences**.

1. **[1pts]** Run the LinUCB algorithm without adding any new arms. Run the algorithm for 5 different seeds. Report the mean and standard deviation of the total fraction correct. You can do this by running the following command:

```
python hw5.py -s 0 1 2 3 4 -u none
```

2. **[2pts]** Run the LinUCB algorithm by adding new arms with the **popular** strategy. Run it with 4 different values of $K \in 500, 1000, 2500, 5000$. Run each K for 5 different seeds. Report the mean and standard deviation of the total fraction correct for each K. You can do this by running the following command:

```
python hw5.py -s 0 1 2 3 4 -u popular -k 500 1000 2500 5000
```

Are your results better or worse than the results when you didn't add any new arms? Why do you think this is the case?
Performance is worse. Probably because we are not adding any useful arms and the added complexity of adding new arms is making the problem harder.

3. **[2pts]** Run the LinUCB algorithm by adding new arms with the **corrective** strategy. Run it with 4 different values of $K \in 500, 1000, 2500, 5000$. Run each K for 5 different seeds. Report the mean and standard deviation of the total fraction correct for each K. You can do this by running the following command:

```
python hw5.py -s 0 1 2 3 4 -u corrective -k 500 1000 2500 5000
```

Which arm update strategy is better – corrective or popular? Or are they the same? Why do you think this is the case?
Performance is exactly the same. Any explanation of why this may be the case is acceptable. One explanation could be that the arm recommended the most is what is recommended incorrectly as well, so both methods end up creating the exact same arms at each step.

4. **[5pts]** Run the LinUCB algorithm by adding new arms with the **counterfactual** strategy. Run it with 4 different values of $K \in 500, 1000, 2500, 5000$. Run each K for 5 different seeds. Report the mean and standard deviation of the total fraction correct for each K. You can do this by running the following command:

```
python hw5.py -s 0 1 2 3 4 -u counterfactual -k 500 1000 2500 5000
```

Plot a table to compare the results from all 3 arm update strategies and when you don't add new arms for different values of K. Which arm update strategy is the best of the three? Which of them perform better than the case where we don't add new arms? Why do you think this is the case?
Results for parts 1,2,3,4 given below. Counterfactual is the best and is the only one that performs better than adding no arms. Any explanation of why this is the case is acceptable. One example could be that it is solving a more structured optimization problem and creating a new arm as opposed to just averaging out existing arms.

K	None	Popular	Corrective	Counterfactual
500	0.569 $\pm$0.077	0.444 $\pm$0.093	0.444 $\pm$0.093	0.705 $\pm$0.035
1000	0.569 $\pm$0.077	0.491 $\pm$0.089	0.491 $\pm$0.089	0.674 $\pm$0.046
2500	0.569 $\pm$0.077	0.534 $\pm$0.083	0.534 $\pm$0.083	0.629 $\pm$0.063
5000	0.569 $\pm$0.077	0.554 $\pm$0.079	0.554 $\pm$0.079	0.572 $\pm$0.087

Table 1: Total Fraction Correct

5. **[2pts]** Now run all the algorithms together for $K = 500$ and plot a graph of the fraction incorrect over time. You can run this with the following command:

```
python hw5.py -s 0 1 2 3 4 -u none popular corrective counterfactual -k 500
--plot-u
```

How do the different strategies perform for $K = 500$?

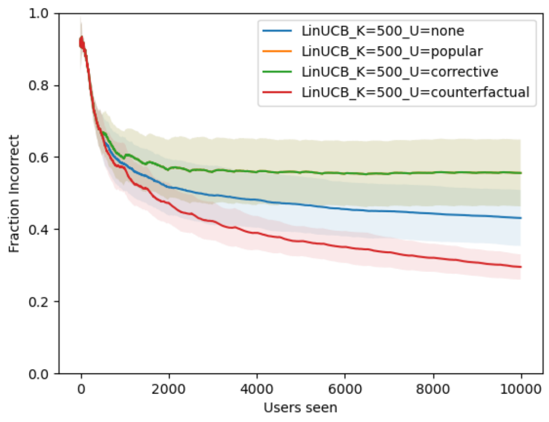

Figure 1: Fraction Incorrect

Again counterfactual is best.

6. **[3pts]** Let us analyse the effect of K on all our algorithms. Plot a graph of the total fraction correct for all the algorithms for $K \in \{100, 250, 500, 1000, 2000, 3000, 4000, 5000\}$. You can run this with the following command:

```
python hw5.py -s 0 1 2 3 4 -u none popular corrective counterfactual -k 100 250
500 1000 2000 3000 4000 5000 --plot-k
```

Which strategies get better as you increase K and which strategies get worse? Why do you think this is the case?

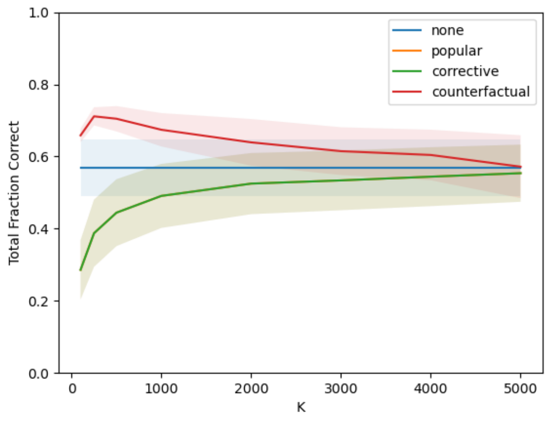

Figure 2: K Analysis

Counterfactual gets worse as K increases, the other two get better. Reasoning is similar to parts 2,3,4 but any good answer is acceptable.

Discussion [5pts]

7. We now would like you to think about the different interactions between content creators, users, and AI algorithms.

   Please look at the accompanying material in the Assignment release Ed post and consider the following (non-exhaustive) possible scenarios.

   (a) **User preferences are static**: if at time t_1 a user only likes to watch videos about dogs, at time t_{100} he/she/they will still only want to watch videos about dogs. As creators learn more about user preferences, they cooperate in making more of the content that the users prefer and are rewarded in the metrics for doing so. AI algorithms also provide recommendations of content most likely to align with the static user preferences.

   (b) In a second situation, user **meta-preferences (e.g. for popular content) are static but the AI algorithm can contribute to feedback loops**. For example, assume some users always prefer popular content, but their specific content preferences will shift over time depending on what appears to be most popular in the content they are exposed to. Initially, automated algorithm randomly selects some content to show more frequently. Seeing that it has been viewed more frequently, content creators create more of that content. The automated algorithm then serves it to more people, causing its popularity to increase. Creators specializing in initially popular content are rewarded, causing a "rich get richer" phenomenon, both at the level of individual content items, and for particular creators.

   (c) Consider an example: users who did not previously like cereal are influenced by an advertisement or by repeated exposure to low-view-count cereal videos. They come to prefer cereal and will now click on cereal videos in the future, but may be unaware that the algorithm influenced this shift in preferences. In this third scenario, **users'**

preferences are dynamic and are influenced and changed by the content served. This can both influence what later content is recommended by the AI algorithm and which future new content is created by content creators.

(d) A more complicated scenario can be when users may change in response to perceived or actual algorithmic classification. For example, a user may click on many items relating to Taylor Swift. This may result in an AI algorithm serving them many future items related to Taylor Swift or implicitly categorizing them as "a Taylor Swift fan." Although the user had not previously realized their own preference for Taylor Swift, once the user sees how much Taylor Swift content they are now being served, the user realizes that the AI algorithm models them as someone who really likes Taylor Swift. The user may react to this perceived labeling of being a "Taylor Swift fan", potentially changing their own behavior either to embrace this classification (and click more on Taylor Swift content) or rejecting this categorization (and click less on future Taylor Swift content). Here users' preferences are dynamic and **users react to their algorithmic** funneling into different soft categories, creating "looping kinds".

[2pts] i. Consider the four situations (of many others) above. Which of these do you think may have problematic ethical implications? What information would be needed to understand which situation is happening in a particular platform-users-content creator universe? Do you think it is likely that platform creators or content creators have access to this information in their standard logs, or do you expect specific experiments would be needed? (2-4 sentences)

[3pts] ii. Who is responsible for identifying a negative feedback loop and intervening to break it: content creators or platforms? Justify your answer with a 3-6 sentence explanation that references one or more of the above scenarios.

i. 1 pt. Identifies at least one of the scenarios as having potential ethical concerns/problematic implications.
1 pt. Identifies information needed to understand which scenario is happening, explains why this information would be helpful in distinguishing scenarios, *and* indicates whether platform or content creators have access to this information. It is not sufficient to say e.g. "You would need access to viewing numbers" without explaining why this would help a content creator identify which scenario they are in.
ii. 1 pt. Identifies a responsible party.
2 pts. Justifies why that party is responsible, or more responsible than the other party, with a reason that answers the question of *why* and does not just restate *that* they are responsible. A partial credit answer (1/2 possible points) is less clear, specific, or plausible than other answers, or fails to give a reason why that *justifies* responsibility. For example, "The platform is responsible because they are ultimately in charge of what happens on YouTube" begins to give an answer, but isn't clear about what the platform is in charge of and doesn't explain why "being in charge" means that they are responsible for this specific outcome.

References

R. D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72(1-2):173–215, Jan. 1995. doi: 10.1016/0004-3702(94)00005-l. URL https://doi.org/10.1016/0004-3702(94)00005-l.

A. Christin. Counting clicks: Quantification and variation in web journalism in the united states and france. *American Journal of Sociology*, 123(5):1382–1415, Mar. 2018. doi: 10.1086/696137. URL https://doi.org/10.1086/696137.

A. Christin and R. Lewis. The drama of metrics: Status, spectacle, and resistance among YouTube drama creators. *Social Media Society*, 7(1):205630512199966, Jan. 2021. doi: 10.1177/2056305121999660. URL https://doi.org/10.1177/2056305121999660.

P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manag. Inf. Syst.*, 6(4):13:1–13:19, 2016. doi: 10.1145/2843948. URL https://doi.org/10.1145/2843948.

J. Kawale and E. Chow. A multi-armed bandit framework for recommendations at netflix. 2018. URL https://info.dataengconf.com/hubfs/SF%2018%20-%20Slides/DS/A%20Multi-Armed%20Bandit%20Framework%20for%20Recommendations%20at%20Netflix.pdf.

L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

R. Mehrotra. Recommendations in a marketplace: Personalizing explainable recommendations with multi-objective contextual bandits. 2018. URL https://mlconf.com/sessions/recommendations-in-a-marketplace-personalizing-explainable-recommendations-with-multi-objective-contextual-bandits/.

A. Swaminathan, A. Krishnamurthy, A. Agarwal, M. Dudík, J. Langford, D. Jose, and I. Zitouni. Off-policy evaluation for slate recommendation, 2017.