IIC3675: Bandits

Introducción

En esta tarea replicaremos algunos de los experimentos mostrados en el capítulo 2 del libro: "*Reinforcement Learning: An introduction.*" Para ello, te proveemos un código base que tiene implementado el *10-armed Testbed* discutido en la sección 2.3 del libro.

El código incluye la clase `BanditEnv` que crea un problema *10-armed bandit*. Al crear un objeto `BanditEnv`, las recompensas esperadas de cada arm son muestreadas usando una distribución normal con media μ y varianza 1, donde μ es dado en el constructor (su valor por defecto es cero).

`BanditEnv` cuenta con tres métodos principales:

- `action_space`: Es un property que retorna el número de arms del bandit (por defecto son 10).
- `step(action)`: Es un método que recibe el *id* del arm a ejecutar. Los ids van del 0 al k-1 para un k-armed bandit. Retorna la recompensa obtenida luego de ejecutar la acción entregada. La recompensa es muestreada de una normal donde la media depende del arm seleccionado y la varianza es 1.
- `best_action`: Es un property que retorna el id del brazo que genera la mayor recompensa (en valor esperado). Se utiliza para analizar cuántas veces el agente ejecutó la acción óptima.

El código también incluye un `Main.py` que contiene el esqueleto de un método que corre 2000 experimentos de 10-armed bandits utilizando un agente que selecciona acciones al azar:

```
NUM_OF_RUNS = 2000
NUM_OF_STEPS = 1000

results = BanditResults()
for run_id in range(NUM_OF_RUNS):
    bandit = BanditEnv(seed=run_id)
    num_of_arms = bandit.action_space
    agent = RandomAgent(num_of_arms)
    best_action = bandit.best_action
    for _ in range(NUM_OF_STEPS):
        action = agent.get_action()
        reward = bandit.step(action)
        agent.learn(action, reward)
        is_best_action = action == best_action
        results.add_result(reward, is_best_action)
    results.save_current_run()

show_results(results)
```

Este código funciona para cualquier agente que herede de `BaseAgent` e implemente estos dos métodos:

- `get_action()`: Retorna la siguiente acción a ejecutar.
- `learn(action, reward)`: Recibe la acción que acaba de ser ejecutada y la recompensa obtenida. Con esa información, se espera que el agente aprenda a tomar mejores decisiones.

El código del `BaseAgent` y `RandomAgent` se encuentran dentro de la carpeta *agents* en el código base.

Instrucciones

Esta tarea puede ser resuelta en pareja o de manera individual.

Debes subir un informe junto con tu código. El código debe incluir un README con instrucciones claras que permitan replicar los experimentos que realizaste. El informe debe incluir gráficos con los experimentos realizados y responder las siguientes preguntas (**justifica tus respuestas**):

a) **[2 puntos]** Implementa la versión incremental del *simple bandit algorithm* (sección 2.4 del libro). Luego replica el experimento de la Figura 2.2. Pon ambos gráficos en tu informe (el que muestra la recompensa promedio y el que muestra el porcentaje de veces que la acción óptima fue seleccionada). ¿Los resultados que obtuviste fueron los esperados? Si la respuesta es *no*, ¿por qué?

b) **[1 punto]** El gráfico de la Figura 2.2 del libro muestra que un algoritmo que usa $\epsilon = 0{,}1$ elige la acción óptima cerca del 80 % de las veces (luego de 1000 pasos de entrenamiento). Esto resulta curioso considerando que usar $\epsilon = 0{,}1$ significa elegir una acción aleatoria el 10 % de las veces. ¿Por qué entonces el rendimiento del agente no es un 90 % al utilizar $\epsilon = 0{,}1$?

c) **[2 puntos]** Implementa la variación del *simple bandit algorithm* donde el *step size* es un valor fijo (sección 2.5 del libro). Luego replica el experimento de la figura 2.3 del libro – donde se muestra la ventaja de utilizar una inicialización optimista. Pon el gráfico resultante en tu informe. ¿Los resultados que obtuviste fueron los esperados? Si la respuesta es *no*, ¿por qué?

d) **[1 punto]** Al utilizar una inicialización optimista con $\epsilon = 0$, existe una repentina subida de rendimiento al inicio del gráfico (seguido por una bajada brusca).[1] ¿A qué se debe esta subida y bajada repentina?

e) **[1 punto]** ¿Por qué en el experimento de inicialización optimista (con $\epsilon = 0$ y $\alpha = 0{,}1$), el agente solo selecciona la acción óptima cerca del 85 % de las veces y no llega a valores cercanos al 100 %?

f) **[2 puntos]** Implementa el *gradient bandit algorithm* (sección 2.8 del libro). Luego replica el experimento de la figura 2.5 del libro. Notar que este experimento utiliza $\mu = 4$ para muestrear los valores esperados de cada arm. Pon el gráfico resultante en tu informe. ¿Los resultados que obtuviste fueron los esperados? Si la respuesta es *no*, ¿por qué?

g) **[1 punto]** ¿Qué hubiese ocurrido en el experimento anterior si usábamos $\mu = 0$ para muestrear los valores esperados de cada arm? ¿cómo usar $\mu = 0$ hubiese afectado los resultados de la figura 2.5 del libro? ¿Por qué ocurre esto?

[1] Para más detalles, ver *Exercise 2.6* del libro.