Demo of bandit algorithms

This is a simple demo of a simulated bandit environment and some baseline algorithms. In this environment, the agent can one of the actions and will receive a binary reward signal whose probability is drawn from a Beta distribution. The probabilities of these beta distributions are not known to the agent. Thus, the agents must estimate these probabilities while also trying to maximize their reward.
import torch
import matplotlib.pyplot as plt

class BernoulliBanditEnvironment:
    def __init__(self, n_actions):
        probs = torch.distributions.Beta(1, 2).sample((n_actions,))
        self.reward_dists = [
            torch.distributions.Bernoulli(probs=p)
            for p in probs
        ]

    def get_reward(self, action_index):
        return self.reward_dists[action_index].sample((1,))[0]

    def simulate_step(self, agent):
        action_index = agent.select_action()
        reward = self.get_reward(action_index)
        agent.update(action_index, reward)
        return action_index, reward

    def simulate(self, agent, n_steps, verbosity=1):
        cum_reward = 0
        for i in range(n_steps):
            action_index, reward = self.simulate_step(agent)
            cum_reward += reward
            if verbosity >= 1:
                print(f'
Step = {i+1:02d}, '
                    f'Action = {action_index.item():d}, '
                    f'Reward = {reward.item():.0f}, '
                    f'Cum reward = {cum_reward:.0f}, '
                    f'Avg reward = {cum_reward/(i+1):.2f}, '
                )
        return cum_reward

torch.manual_seed(0)
env = BernoulliBanditEnvironment(3)
print('Reward distributions')
print(env.reward_dists)
print('
Random rewards for first distribution')
print([env.get_reward(0) for i in range(20)])

Reward distributions
[Bernoulli(probs: 0.6865779757499695), Bernoulli(probs: 0.20974847674369812), Bernoulli(probs: 0.16393446923202246)]

Random rewards for first distribution
[tensor(1.), tensor(1.), tensor(0.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(0.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(0.), tensor(1.), tensor(1.), tensor(0.), tensor(0.), tensor(0.), tensor(0.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.), tensor(1.)]
Interactive agent

For this agent, just uncomment the line below to interactively provide the actions yourself and see how well you can do.

```python
In [2]:
   ...: class InteractiveAgent:
   ...:     def __init__(self, n_actions):
   ...:         self.n_actions = n_actions
   ...:
   ...:     def select_action(self):
   ...:         action_index = -1
   ...:         while action_index < 0:
   ...:             print(f'Enter action index from 0-{self.n_actions-1}"
   ...:             try:
   ...:                 action_index = int(input())
   ...:             except Exception:
   ...:                 action_index = -1
   ...:             if action_index >= 0 and action_index < self.n_actions:
   ...:                 break
   ...:             else:
   ...:                 print('Incorrect input, try again."
   ...:                 action_index = -1
   ...:             return torch.tensor(action_index)
   ...:
   ...:     def update(self, action_index, reward):
   ...:         return self
   ...:
   ...:torch.manual_seed(2)
   ...:env = BernoulliBanditEnvironment(2)
   ...:interactive_agent = InteractiveAgent(len(env.reward_dists))
   ...:
   ...:# Uncomment to be interactive
   ...:#env.simulate(interactive_agent, n_steps=2)
```

Oracle agent

This is like a cheating agent that actually already knows the probabilities for each action and thus can choose the optimal action every time.
class OracleAgent:
    def __init__(self, env):
        self.optimal_action = torch.argmax(torch.tensor(
            [dist.probs for dist in env.reward_dists
        ]))

    def select_action(self):
        return self.optimal_action

    def update(self, action_index, reward):
        return self

    def __str__(self):
        return 'Oracle'

torch.manual_seed(2)  # 2 and 5
env = BernoulliBanditEnvironment(2)
oracle = OracleAgent(env)
env.simulate(oracle, n_steps=20)

Step = 01, Action = 1, Reward = 0, Cum reward = 0, Avg reward = 0.00,
Step = 02, Action = 1, Reward = 1, Cum reward = 1, Avg reward = 0.50,
Step = 03, Action = 1, Reward = 1, Cum reward = 2, Avg reward = 0.67,
Step = 04, Action = 1, Reward = 0, Cum reward = 2, Avg reward = 0.50,
Step = 05, Action = 1, Reward = 1, Cum reward = 3, Avg reward = 0.60,
Step = 06, Action = 1, Reward = 1, Cum reward = 4, Avg reward = 0.67,
Step = 07, Action = 1, Reward = 1, Cum reward = 5, Avg reward = 0.71,
Step = 08, Action = 1, Reward = 0, Cum reward = 5, Avg reward = 0.62,
Step = 09, Action = 1, Reward = 1, Cum reward = 6, Avg reward = 0.67,
Step = 10, Action = 1, Reward = 1, Cum reward = 7, Avg reward = 0.70,
Step = 11, Action = 1, Reward = 1, Cum reward = 8, Avg reward = 0.73,
Step = 12, Action = 1, Reward = 1, Cum reward = 9, Avg reward = 0.75,
Step = 13, Action = 1, Reward = 1, Cum reward = 10, Avg reward = 0.77,
Step = 14, Action = 1, Reward = 1, Cum reward = 11, Avg reward = 0.79,
Step = 15, Action = 1, Reward = 1, Cum reward = 12, Avg reward = 0.80,
Step = 16, Action = 1, Reward = 1, Cum reward = 13, Avg reward = 0.81,
Step = 17, Action = 1, Reward = 1, Cum reward = 14, Avg reward = 0.82,
Step = 18, Action = 1, Reward = 0, Cum reward = 14, Avg reward = 0.78,
Step = 19, Action = 1, Reward = 1, Cum reward = 15, Avg reward = 0.79,
Step = 20, Action = 1, Reward = 1, Cum reward = 16, Avg reward = 0.80,

Out[3]: tensor(16.)
Try to see how close you can get to the oracle agent

In [4]:

```python
env = BernoulliBanditEnvironment(2)
n_steps = 10

interactive_agent = InteractiveAgent(len(env.reward_dists))
total_reward = 0

# Uncomment to be interactive
# total_reward = env.simulate(interactive_agent, n_steps=n_steps)
oracle_reward = env.simulate(OracleAgent(env), n_steps=10000, verbosity=0)
```

```
The probabilities were [0.3823629915714264, 0.7332632541656494]

Your reward was $0, the oracle policy would have received an award close to $7.38
```

Random agent

This agent merely chooses an action at random.
Greedy agent

This chooses the best action given it’s current estimate of the action value function. Note the initialization value can matter significantly as higher values will encourage it to explore more.
\textbf{\(\epsilon\)-greedy agent}

With some probability, take a random action otherwise pick greedy action.
In [7]:

class EpsilonGreedyAgent(GreedyAgent):
    def __init__(self, n_actions, epsilon):
        super().__init__(n_actions, init_value=0)
        self.epsilon = epsilon

    def select_action(self):
        if torch.rand((1,))[0] < self.epsilon:
            return torch.randint(len(self.action_value_func), (1,))[0]
        else:
            return torch.argmax(self.action_value_func)

    def __str__(self):
        return f'\$\epsilon\$-Greedy($\epsilon={self.epsilon:.2f}$)'

# Try init_value = 0, 1 or 100
torch.manual_seed(0)
env = BernoulliBanditEnvironment(3)
epsilon_greedy = EpsilonGreedyAgent(len(env.reward_dists), epsilon=0.2)
env.simulate(epsilon_greedy, n_steps=20)

Step = 01, Action = 0, Reward = 1, Cum reward = 1, Avg reward = 1.00,
Step = 02, Action = 0, Reward = 1, Cum reward = 2, Avg reward = 1.00,
Step = 03, Action = 0, Reward = 1, Cum reward = 3, Avg reward = 1.00,
Step = 04, Action = 0, Reward = 1, Cum reward = 4, Avg reward = 1.00,
Step = 05, Action = 0, Reward = 1, Cum reward = 5, Avg reward = 1.00,
Step = 06, Action = 1, Reward = 0, Cum reward = 5, Avg reward = 0.83,
Step = 07, Action = 0, Reward = 0, Cum reward = 5, Avg reward = 0.71,
Step = 08, Action = 0, Reward = 1, Cum reward = 6, Avg reward = 0.75,
Step = 09, Action = 0, Reward = 1, Cum reward = 7, Avg reward = 0.78,
Step = 10, Action = 2, Reward = 0, Cum reward = 7, Avg reward = 0.70,
Step = 11, Action = 0, Reward = 1, Cum reward = 8, Avg reward = 0.73,
Step = 12, Action = 0, Reward = 0, Cum reward = 8, Avg reward = 0.67,
Step = 13, Action = 0, Reward = 0, Cum reward = 8, Avg reward = 0.62,
Step = 14, Action = 0, Reward = 0, Cum reward = 8, Avg reward = 0.57,
Step = 15, Action = 0, Reward = 1, Cum reward = 9, Avg reward = 0.60,
Step = 16, Action = 2, Reward = 0, Cum reward = 9, Avg reward = 0.56,
Step = 17, Action = 0, Reward = 0, Cum reward = 9, Avg reward = 0.53,
Step = 18, Action = 0, Reward = 1, Cum reward = 10, Avg reward = 0.56,
Step = 19, Action = 0, Reward = 1, Cum reward = 11, Avg reward = 0.58,
Step = 20, Action = 0, Reward = 1, Cum reward = 12, Avg reward = 0.60,

Out[7]:  tensor(12.)
Let's compare these algorithms.

```python
# Compare algorithms
n_actions = 100
n_steps = 1000
agents = [
    RandomAgent(n_actions),
    GreedyAgent(n_actions, init_value=0),
    GreedyAgent(n_actions, init_value=1),
    EpsilonGreedyAgent(n_actions, epsilon=0.0),
    EpsilonGreedyAgent(n_actions, epsilon=0.01),
    EpsilonGreedyAgent(n_actions, epsilon=0.1),
]
labeled_list = []
mean_reward_tensor = []
for seed in range(30):
    mean_reward_list = []
    torch.manual_seed(seed)
    env = BernoulliBanditEnvironment(n_actions)
    #print([dist.probs.item() for dist in env.reward_dists])
    for agent in agents + [OracleAgent(env)]:
        torch.manual_seed(seed * 1000)
        mean_reward = []
        cum_reward = 0
        for step in range(n_steps):
            _, reward = env.simulate_step(agent)
            cum_reward += reward
            mean_reward.append(cum_reward / (step + 1))
        mean_reward_list.append(mean_reward)
        if seed == 0:
            labeled_list.append(str(agent))
        mean_reward_tensor.append(mean_reward_list)

# Plot averages
average_reward_agent = torch.tensor(mean_reward_tensor).mean(dim=0)
fig = plt.figure(figsize=(6, 4), dpi=100)
for x, label in zip(average_reward_agent, labeled_list):
    plt.plot(x, label=label)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

Out[8]: `<matplotlib.legend.Legend at 0x7fa5b7ed73a0>`