Оглавление

6 Задача о многоруком бандите

В предыдущих главах были представлены фундаментальные концепции **обучения с подкреплением (RL)** и некоторые алгоритмы RL, а также моделирование RL-задач в форме **марковского процесса принятия решений (MDP)**. Также были описаны различные алгоритмы (с моделью и без модели), используемые для решения MDP. В этой главе будет рассмотрена одна из классических задач RL — **задача о многоруком бандите**, или **MAB** (multi-armed bandit). Вы узнаете, что собой представляет эта задача, как она решается при помощи разных алгоритмов и как выбрать наиболее подходящий рекламный баннер, который будет получать наибольшее число переходов, средствами MAB. Также будет описан так называемый контекстный бандит, широко применяемый при построении рекомендационных систем.

В этой главе рассматриваются следующие темы:

- Задача о многоруком бандите (MAB).
- Эпсилон-жадный алгоритм.
- Алгоритм softmax-исследования.
- Алгоритм верхней границы доверительного интервала.
- Алгоритм выборки Томпсона.
- Практические применения MAB.
- Выбор подходящего рекламного баннера с использованием MAB.
- Контекстные бандиты.

Задача MAB

Задача о многоруком бандите (*MAB*) — одна из классических задач в RL. Многорукий бандит — игровой автомат; азартная игра, в которой игрок нажимает на рычаг и получает награду (выигрыш) на основании случайно сгенерированного распределения вероятностей. Отдельный автомат называется «одноруким бандитом»; если же таких автоматов несколько, это называется «многоруким бандитом» или k-руким бандитом.

Многорукий бандит выглядит так:

Так как каждый автомат дает выигрыш из собственного вероятностного распределения, наша цель — определить, какой автомат принесет максимальную накапливаемую награду за конкретный промежуток времени. Таким образом, на каждом временном кванте t агент выполняет действие a_t, то есть нажимает рычаг игрового автомата и получает награду r_t, при этом цель агента заключается в максимизации накапливаемой награды.

Определим ценность руки $Q(a)$ как среднюю награду, получаемую при нажатии рычага:

$$Q(a) = \frac{\text{Сумма наград, полученных на руке}}{\text{Общее количество активизаций руки}}.$$

Таким образом, *оптимальной* будет называться рука, обеспечивающая максимальную накапливаемую награду:

$$Q(a^*) = \text{Max}Q(a).$$

Цель агента — найти оптимальную руку и минимизировать потери, которые могут определяться как затраты на получение информации о том, какая из k рук является оптимальной. Как найти лучшую руку? Исследовать все руки или выбрать ту руку, которая уже дает максимальную накапливаемую награду? Мы снова сталкиваемся с дилеммой компромисса между исследованием и эксплуатацией. В этой главе будет показано, как эта дилемма решается с помощью различных стратегий исследования:

- эпсилон-жадная стратегия;
- softmax-исследование;
- алгоритм верхней границы доверительного интервала;
- алгоритм выборки Томпсона.

Прежде чем двигаться дальше, установите среды `bandit` в OpenAI Gym; для этого введите в терминале следующие команды:

```
git clone https://github.com/JKCooper2/gym-bandits.git
cd gym-bandits
pip install -e .
```

После установки импортируйте `gym` и `gym_bandits`:

```
import gym_bandits
import gym
```

Теперь нужно инициализировать среду; мы будем использовать MAB с десятью руками:

```
env = gym.make("BanditTenArmedGaussian-v0")
```

Размер пространства действий равен 10, так как у бандита 10 рук. Соответственно, команда

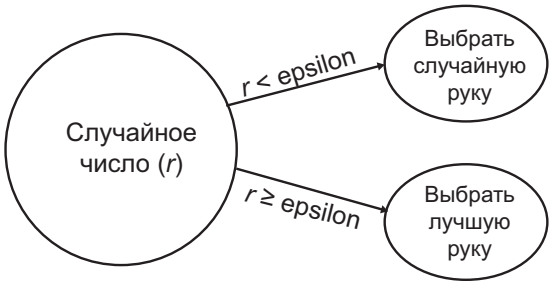
```
env.action_space
```

выводит следующий результат:

```
10
```

Эпсилон-жадная стратегия

Вы уже неоднократно сталкивались с эпсилон-жадной стратегией; здесь выбирается либо лучшая рука с вероятностью 1 – epsilon, либо случайная рука с вероятностью epsilon:



Посмотрим, как выбрать лучшую руку с применением эпсилон-жадной стратегии:

1. Инициализировать все переменные:

```
# Количество раундов (итераций)
num_rounds = 20000

# Количество нажатий на рычаг
count = np.zeros(10)

# Сумма наград для каждой руки
sum_rewards = np.zeros(10)

# Q, то есть средняя награда
Q = np.zeros(10)
```

2. Затем определить функцию `epsilon_greedy`:

```
def epsilon_greedy(epsilon):
    rand = np.random.random()
    if rand < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q)
    return action
```

3. Перейти к нажатию рычагов:

```
for i in range(num_rounds):
    # Выбрать руку с применением эпсилон-жадной политики
    arm = epsilon_greedy(0.5)
    # Получить награду
    observation, reward, done, info = env.step(arm)
    # Обновить счетчик для этой руки
    count[arm] += 1
    # Включить в сумму наград от руки
    sum_rewards[arm]+=reward
    # Вычислить Q - среднюю награду от руки
    Q[arm] = sum_rewards[arm]/count[arm]

print( 'The optimal arm is {}'.format(np.argmax(Q)))
```

Результат выполнения:

```
The optimal arm is 3
```

Алгоритм softmax-исследования

Softmax-исследование, также называемое *исследованием Больцмана*, — еще одна стратегия, применяемая для нахождения оптимальной руки. В эпсилон-жадной стратегии все не лучшие руки считаются равноправными, а в softmax-исследовании рука выбирается на основании вероятности из распределения Больцмана. Эта вероятность определяется следующей формулой:

$$P_t(a) = \frac{\exp(Q_t(a)/\tau)}{\sum_{i=1}^{n}\exp(Q_t(i)/\tau)}.$$

Здесь τ — температурный коэффициент, который определяет, сколько случайных рук можно исследовать. При высоких значениях τ все руки будут исследоваться с равной вероятностью, а при низких значениях τ будут выбираться руки с высокой наградой. Последовательность действий выглядит так:

1. Инициализация переменных:

```
# Количество раундов (итераций)
num_rounds = 20000

# Количество нажатий на рычаг
count = np.zeros(10)
```

```
# Сумма наград для каждой руки
sum_rewards = np.zeros(10)

# Q, то есть средняя награда
Q = np.zeros(10)
```

2. Определение softmax-функции:

```
def softmax(tau):
    total = sum([math.exp(val/tau) for val in Q])
    probs = [math.exp(val/tau)/total for val in Q]
    threshold = random.random()
    cumulative_prob = 0.0
    for i in range(len(probs)):
        cumulative_prob += probs[i]
        if (cumulative_prob > threshold):
            return i
    return np.argmax(probs)
```

3. Переход к нажатию рычагов:

```
for i in range(num_rounds):
    # Выбрать руку с применением softmax-политики
    arm = softmax(0.5)
    # Получить награду
    observation, reward, done, info = env.step(arm)
    # Обновить счетчик для этой руки
    count[arm] += 1
    # Просуммировать награды от руки
    sum_rewards[arm]+=reward
    # Вычислить Q - среднюю награду от руки
    Q[arm] = sum_rewards[arm]/count[arm]
print( 'The optimal arm is {}'.format(np.argmax(Q)))
```

Результат:

```
The optimal arm is 3
```

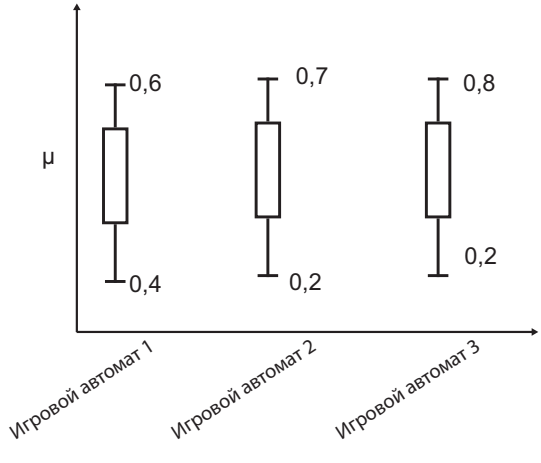
Алгоритм верхней границы доверительного интервала

Эпсилон-жадные и softmax-алгоритмы обеспечивают исследование случайных действий с определенной вероятностью. Исследование разных вариантов может привести к опробованию действий, вообще не несущих хорошей награды.

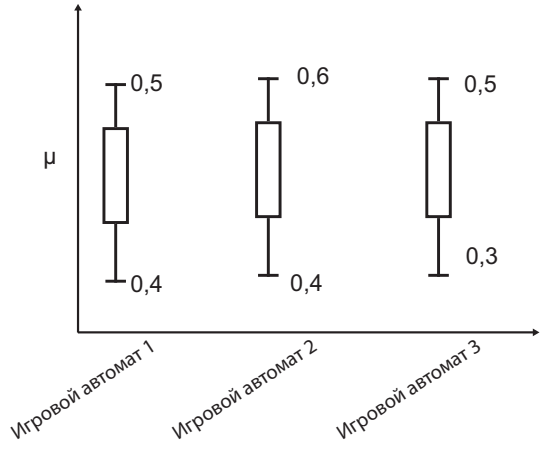
При этом не хотелось бы упустить те хорошие руки, которые принесли низкую награду в начальных итерациях. Для решения таких проблем существует алгоритм *верхней границы доверительного интервала* (*UCB*, Upper Confidence Bound), основанный на принципе оптимизма перед лицом неопределенности.

Алгоритм UCB помогает выбрать лучшую руку на основании доверительного интервала. Что это такое? Допустим, есть две руки. Мы опробуем оба варианта; первая рука обеспечивает награду 0,3, а вторая — 0,8. Тем не менее по одной попытке нельзя прийти к выводу, что рука 2 обеспечивает более высокую награду. Нужно попробовать нажать на рычаг несколько раз, вычислить среднее значение награды, полученной для каждой руки, и выбрать вариант с более высоким средним значением. Но как определить правильное среднее значение для каждой из этих рук? На помощь приходит *доверительный интервал* — интервал, в котором лежит средняя награда для руки. Если доверительный интервал руки 1 имеет вид [0,2, 0,9], то это означает, что среднее значение для руки 1 лежит в этом интервале. Значение 0,2 называется *нижней границей доверительного интервала*, а 0,9 называется *верхней границей доверительного интервала*, или *UCB*. Алгоритм UCB выбирает для исследования игровой автомат с более высоким значением UCB.

Допустим, есть три игровых автомата, и на каждом из автоматов вы сыграли 10 раз. Доверительные интервалы для этих трех игровых автоматов изображены на следующем рисунке:



Мы видим, что игровой автомат 3 имеет высокое значение UCB. Тем не менее не стоит делать вывод, что этот игровой автомат обеспечивает хорошую награду, на основании всего 10 попыток. С другой стороны, после нескольких попыток доверительный интервал будет более точным. Таким образом, со временем доверительный интервал сужается к фактическому значению, как показано на следующем рисунке. Итак, мы можем выбрать игровой автомат 2 с высоким значением UCB:



Принцип работы алгоритма UCB очень прост:

1. Выбрать действие (руку) с высокой суммой средней награды и верхней границей доверительного интервала.
2. Нажать на рычаг и получить награду.
3. Обновить награду и границу доверительного интервала для руки.

Но как вычислить UCB?

Для этого можно воспользоваться формулой $\sqrt{\frac{2\log(t)}{N(a)}}$, где $N(a)$ — количество нажатий на рычаг, а t — общее количество раундов.

Итак, в алгоритме UCB рука выбирается по следующей формуле:

$$Arm = \arg\max_a \left[Q(a) + \sqrt{\frac{2\log(t)}{N(a)}} \right].$$