MDP-Основанная Рекомендательная Система

Введение

- Рекомендательные системы уменьшают информационную перегрузку
- Последовательность действий пользователя важна
- MDP учитывает долгосрочные эффекты

Основные понятия

Марковский процесс — это процесс, для которого можно сделать прогноз относительно будущих результатов, основываясь исключительно на его текущем состоянии

Основные понятия

Цепь Маркова — это разновидность марковского процесса, который имеет либо дискретное пространство состояний, либо дискретный набор индексов

$$\mathbb{P}(X_{n+1} = i_{n+1} \mid X_n = i_n, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = \mathbb{P}(X_{n+1} = i_{n+1} \mid X_n = i_n).$$

Цепь Маркова называется **однорóдной**, если матрица переходных вероятностей не зависит от номера шага, то есть

$$P_{ij}(n) = P_{ij}, \quad \forall n \in \mathbb{N}.$$

Основные понятия

Марковский процесс принятия решений (MDP) представляет собой модель последовательного принятия решений в условиях неопределённости результатов

Марковский процесс принятия решений по определению представляет собой четверку: S,A,Rwd,tr, где S — множество состояний, A — множество действий, Rwd — функция вознаграждения, которая присваивает реальное значение каждой паре «состояние/действие», а tr — функция перехода между состояниями, которая определяет вероятность перехода между каждой парой состояний при выполнении каждого действия

стационарная политика для модели «множество состояний — действие» π — это отображение состояний в действия, определяющее, какое действие следует выполнять в каждом состоянии. При наличии такой оптимальной политики π на каждом этапе процесса принятия решения программе нужно лишь определить, в каком состоянии s он находится, и выполнить действие a = π(s).

Алгоритм Value Iteration

$$V_{t+1}(s) = \max_{a \in A} \left[Rwd(s,a) + \gamma \sum tr(s,a,s_j) V_t(s_j) \right]$$

$$\pi^*(s) = \arg\max_a \left[Rwd(s,a) + \gamma \sum_{s_j} tr(s,a,s_j) V^*(s_j) \right]$$

Алгоритм Policy Iteration

$$V^{\pi}(s) = Rwd(s,\pi(s)) + \gamma \sum_{s_j \in S} tr(s,\pi(s),s_j) V^{\pi}(s_j)$$

- $Rwd(s, \pi(s))$

  мгновенное вознаграждение за действие, которое стратегия выбирает в состоянии s.

- γ

  коэффициент дисконтирования (от 0 до 1).

  Определяет, насколько важны будущие награды.

- $tr(s, a, s_j)$

  вероятность того, что после выполнения действия a в состоянии s система перейдёт в состояние s_j.

  Это **вероятностная функция перехода**, часть MDP.

- $\sum_{s_j} tr(s, \pi(s), s_j) V^{\pi}(s_j)$

  ожидаемая ценность следующего шага, учитывающая все возможные состояния, куда можно попасть.

$$\begin{aligned} V_i(s) &= Rwd(s,\pi_i(s)) + \gamma \sum_{s_j \in S} tr(s,\pi_i(s),s_j) V_i(s_j), \\ \pi_{i+1}(s) &= \operatorname*{argmax}_{a \in A} \left[Rwd(s,a) + \gamma \sum_{s_j \in S} tr(s,a,s_j) V_i(s_j) \right]. \end{aligned}$$

Вычислительная сложность

1. **Скорость сходимости:**
– Итерация значения обычно сходится быстрее с точки зрения вычислительных шагов, поскольку она обновляет функцию значения непосредственно с использованием уравнения оптимальности Беллмана. Однако каждая итерация включает в себя операцию максимизации всех действий, что может потребовать больших вычислительных затрат.
– Итерация политики требует меньшего числа итераций. Однако каждый шаг оценки политики включает в себя решение системы линейных уравнений, которая может требовать больших вычислительных ресурсов.

2. **Вычислительная сложность:**
– Итерация значения имеет временную сложность $o(S^2|A|)$ за итерацию, где S количество состояний и A это количество действий.
– Итерация политики имеет временную сложность $o(S^3)$ для оценки политики (при условии прямого решения линейных уравнений) и $o(S^2|A|)$ для улучшения политики

Как инициализировать функцию переходов?

метод максимального правдоподобия: $tr_{MC}(\langle x_1, x_2, x_3 \rangle, \langle x_2, x_3, x_4 \rangle) = \frac{count(\langle x_1, x_2, x_3, x_4 \rangle)}{count(\langle x_1, x_2, x_3 \rangle)}$

где $count(x_1, x_2, \ldots, x_k)$ — количество раз, которое последовательность $x_1, x_2, \ldots, x_k$ встречалась в наборе данных.

Однако такая модель функции перехода страдает от проблемы разреженности данных и плохо работает на практике.

Как улучшить оценку?
Skipping модель

После того как мы посчитали обычные переходы между состояниями (основанные на реально наблюдённых последовательных шагах пользователя), мы добавляем *дробные* счётчики для тех переходов, которые могли бы произойти, если пользователь «перепрыгнул» через несколько элементов последовательности.

Пусть у пользователя есть последовательность действий:

$$x_1, x_2, \ldots, x_n.$$

Тогда для каждого фрагмента длины 3:

$$(x_i, x_{i+1}, x_{i+2}),$$

мы рассматриваем **возможность пропуска** и добавляем небольшие дробные веса к переходам:

$$(x_i, x_{i+1}, x_{i+2}) \longrightarrow (x_{i+1}, x_{i+2}, x_j),$$

для всех позиций j, которые находятся *дальше*, то есть удовлетворяют:

$$i + 3 < j \leq n.$$

Вес этого «пропущенного» перехода задаётся формулой:

$$\Delta = \frac{1}{2^{j-(i+3)}}.$$

Нормализация – $$tr_{MC}(s, s') = \frac{count(s, s')}{\sum_{s'} count(s, s')}$$

Как улучшить оценку?

Второе усовершенствование — это форма кластеризации. Этот подход, основанный на свойствах нашей предметной области, использует сходство последовательностей. Например, состояния x, y, z и w, y, z похожи, потому что некоторые элементы, присутствующие в первом, присутствуют и во втором. Суть подхода заключается в том, что вероятность перехода из состояния s в состояние s' можно предсказать по переходам из состояния t в состояние s' , где состояния s и t схожи.

В частности, мы определяем сходство состояний si и sj

как $$sim(s_i, s_j) = \sum_{m=1}^{k} \delta(s_i^m, s_j^m) \cdot (m+1)$$

Затем определяем количество совпадений между состояниями s и s'

как $$simcount(s, s') = \sum_{s_i} sim(s, s_i) \cdot tr_{MC}^{old}(s_i, s')$$

Тогда новая вероятность перехода из состояния s в состояние s' определяется следующим образом

$$tr_{MC}(s, s') = \frac{1}{2} tr_{MC}^{old}(s, s') + \frac{1}{2} \frac{simcount(s, s')}{\sum_{s''} simcount(s, s'')}$$

Численный пример

Численный пример представлен в латех файле

Код

```
def policy_evaluation_exact(policy, tr, R, gamma=0.9):
    S = tr.shape[0]
    Ppi = np.zeros((S, S))
    Rpi = np.zeros(S)

    for s in range(S):
        a = policy[s]
        Ppi[s] = tr[s, a]
        Rpi[s] = R[s, a]

    A = np.eye(S) - gamma * Ppi
    V = np.linalg.solve(A, Rpi)
    return V

def policy_iteration_exact(tr, R, gamma=0.9, max_iter=50):
    S, A, _ = tr.shape
    # Initial greedy policy by immediate reward
    policy = np.argmax(R, axis=1).astype(int)
    stable = False

    while not stable:
        V = policy_evaluation_exact(policy, tr, R, gamma)
        stable = True
        for s in range(S):
            Q = R[s] + gamma * (tr[s] @ V)
            best = np.argmax(Q)
            if best != policy[s]:
                policy[s] = best
                stable = False

    return policy, V  # fallback
```

Результат:

Optimal policy:

s1 -> a2

s2 -> a2

s3 -> a1

Optimal value function V*:

V(s1) = 12.3988

V(s2) = 12.8430

V(s3) = 13.1589

Источники

Shani G., Brafman R.I., Shimony S.E. Model-based online learning of POMDPs // *Journal of Machine Learning Research*. Vol. 6. P. 1649-1681. https://www.jmlr.org/papers/volume6/shani05a/shani05a.pdf

Марковский процесс принятия решений [Электронный ресурс] // *Википедия: Свободная энциклопедия*. — 2024. — 31 октября. — URL: https://en.wikipedia.org/wiki/Markov_decision_process

Цепь Маркова [Электронный ресурс] // *Википедия: Свободная энциклопедия*. — 2024. — 21 ноября. — URL: https://en.wikipedia.org/wiki/Markov_chain