Project-2 Blackjack 报告：基于 MDP 的人工智能

陈笑宇 21340246003　林子开 21307110161

2023 年 11 月 13 日

摘要　本报告包含了本次 Blackjack 作业的所有书面部分，并对本次作业的编程部分进行了简要介绍，以帮助读者理解编程部分。**在问题一中**，我们对一个简单的问题进行了价值迭代，并给出了详细的过程与相应的最优值函数和最优策略。**在问题二中**，我们分别对在转移函数中添加噪声的 MDP，非循环的 MDP，以及用 $\gamma = 1$ 的 MDP 求解器求解 $\gamma < 1$ 的 MDP，共三个问题进行了分析。**在问题三中**，我们根据 Blackjack 的游戏规则，将 Blackjack 实现为 MDP，并且设计出了一套偷看率大于等于 10% 的牌。**在问题四中**，我们使用函数逼近的 Q-learning 求解 Blackjack 问题，并与价值迭代的效果进行了对比，随后我们改进了特征提取器使之能够接近价值迭代的表现，最后我们说明了当规则变化时，从原 MDP 所得到的策略不再是最优策略，得到的平均回报会显著下降。

目录

1　问题 1：手动价值迭代

1.1　值迭代

到达各个状态的奖励如表1所示：

表 1: 到达各状态的奖励

s	-2	-1	0	1	2
$r(s)$	20	-5	-5	-5	100

转移的条件概率如表2所示：

表 2: 转移的条件概率

	$\mathrm{T}(s'\|s,a=-1)$	$\mathrm{T}(s'\|s,a=1)$
$s'=s+1$	0.2	0.3
$s'=s+1$	0.8	0.7

计算奖励函数的公式为：

$$R(s,a)=\mathbb{E}[r_t|s_t=s,a_t=a]=\sum_{s'}\mathrm{T}(s'|s,a)r(s')$$

奖励函数 $R(s,a)$ 的计算结果如下表3所示，该表的结果不随迭代而改变。

表 3: 奖励函数

s	$R(s,a=-1)$	$R(s,a=+1)$
-2	–	–
-1	15	12.5
0	-5	-5
$+1$	16	26.5
$+2$	–	–

值迭代公式为：

$$V_{k+1}(s)=\max_{a\in\mathcal{A}(s)}\left\{R(s,a)+\gamma\sum_{s'}\mathrm{T}(s'|s,a)V_k(s')\right\}$$

在本小节中，统一取 $\gamma=1$.

初始时，设置所有状态的 V 值为 0：

表 4: 初始状态的 V 值

s	-2	-1	0	$+1$	$+2$
$V_0(s)$	0	0	0	0	0

基于 $V_0(s)$ 得到的最优策略为：

表 5: 基于 $V_0(s)$ 的最优策略 π_0^*

s	-2	-1	0	$+1$	$+2$
$\pi_0(s)$	–	-1	$-1,+1$	$+1$	–

进行第一次值迭代后各个状态的 V 值如下：

表 6: 迭代第 1 次后的 V 值

s	-2	-1	0	$+1$	$+2$
$V_1(s)$	0	15	-5	26.5	0

基于 $V_1(s)$ 得到的最优策略为：

表 7: 基于 $V_1(s)$ 的最优策略 π_1^*

s	-2	-1	0	$+1$	$+2$
$\pi_1(s)$	$-$	-1	$+1$	$+1$	$-$

进行第 2 次值迭代后各个状态的 V 值如下：

表 8: 迭代第 2 次后的 V 值

s	-2	-1	0	$+1$	$+2$
$V_2(s)$	0	14	13.45	23	0

1.2 基于两轮值迭代的最优策略 π_2^*

对于非终止状态 s 而言，迭代 k 轮后的最优策略可以表示为：

$$\pi_k^*(s) = a_k(s) = \underset{a \in \mathcal{A}(s)}{\arg\max} \left\{ R(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a) V_k(s') \right\}$$

当 $s=-1, a=-1$ 时，

$$V = 15 + 0.8 \times 0 + 0.2 \times 13.45 = 17.69$$

当 $s=-1, a=+1$ 时，

$$V = 12.5 + 0.7 \times 0 + 0.3 \times 13.45 = 16.535$$

因此，$s=-1$ 时，$\pi_2^*(s)|_{s=-1} = -1$

当 $s=0, a=-1$ 时，

$$V = -5 + 0.8 \times 14 + 0.2 \times 23 = 10.8$$

当 $s=0, a=+1$ 时，

$$V = -5 + 0.7 \times 14 + 0.3 \times 23 = 11.7$$

因此，$s=0$ 时，$\pi_2^*(s)|_{s=0} = +1$

当 $s=+1, a=-1$ 时，

$$V = 16 + 0.8 \times 13.45 + 0.2 \times 0 = 26.76$$

当 $s=+1, a=+1$ 时，

$$V = 26.5 + 0.7 \times 13.45 + 0.3 \times 0 = 35.915$$

因此，$s=+1$ 时，$\pi_2^*(s)|_{s=+1} = +1$

将基于 $V_2(s)$ 的最优策略 $\pi_2^*(s)$ 列表如下：

表 9: 基于 $V_2(s)$ 的最优策略 π_2^*

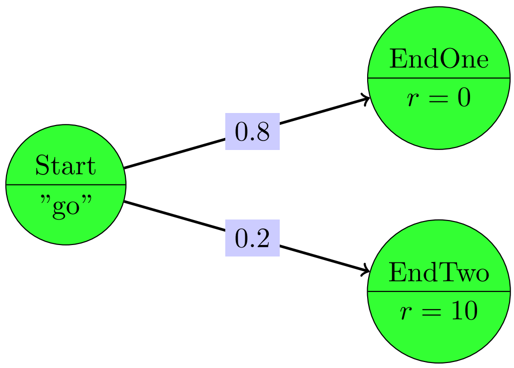
s	-2	-1	0	$+1$	$+2$
$\pi_2(s)$	$-$	-1	$+1$	$+1$	$-$

2 问题 2: 分析并求解某些特殊的 MDP 问题

2.1 在转移函数中添加噪声后最优值的变化

我们已经在 `submission.py` 文件中的 `CounterexampleMDP` 部分构造出了一个反例。下面对这个反例进行简要介绍。

我们设置起始状态为“Start”，起始状态时能采取的动作为“go”，采取“go”后，有 0.8 的概率前往终止状态“EndOne”，有 0.2 的概率前往终止状态“EndTwo”。到达“EndOne”的奖励为 0，到达“EndTwo”的奖励为 10。此外，设置 $V(\text{EndOne})=0, V(\text{EndTwo})=0$。示意图如下：



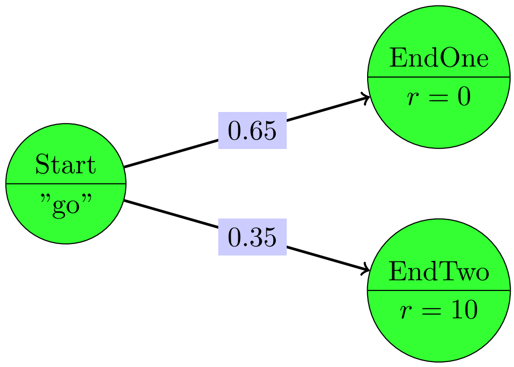
此时

$$V_1(S_{\text{Start}}) = R_1(S = S_{\text{Start}}, a = \text{"go"}) = 0.8 \times 0 + 0.2 \times 10 = 2$$

在转移函数中添加噪声，此时转移的条件概率为：

$$\mathrm{T}(S = \text{EndOne}|S = S_{\text{Start}}, a = \text{"go"}) = 0.5 \times 0.8 + 0.25 = 0.65$$
$$\mathrm{T}(S = \text{EndTwo}|S = S_{\text{Start}}, a = \text{"go"}) = 0.5 \times 0.2 + 0.25 = 0.35$$

示意图如下:



此时

$$V_2(S_{\text{Start}}) = R_2(S = S_{\text{Start}}, a = \text{"go"}) = 0.65 \times 0 + 0.35 \times 10 = 3.5$$

我们可以发现 $V_1(S_{\text{Start}}) < V_2(S_{\text{Start}})$，因此原命题不成立。

2.2 非循环 MDP 最优值的高效算法：基于动态规划思想

对于非循环 MDP 而言，由于访问过的状态不会再次被访问到，因此，在最优值函数中：

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \left\{ R(s,a) + \gamma \sum_{s'} \mathrm{T}(s'|s,a) V^*(s') \right\}$$

$V^*(s')$ 一定不包含已经访问过的状态的最优值，只包含那些未访问过的状态的最优值。此外，注意到状态 s 的最优质 $V^*(s)$ 依赖于子问题，也即后序可能访问到的状态 s' 的最优值 $V^*(s')$ 的求解，因此，可以使用**动态规划**的思想，设计高效的算法。

根据动态规划的思想，我们将建立一张作为**全局变量的动态更新的查询表**，这张表将记录每个状态 s 的最优值是否已经被计算出来，在初始化时，$\forall s, V^*(s) \longleftarrow NULL$。此外，我们定义所有终止状态的节点的最优值函数为 $V^*(s) = 0$,if $\{s' : \mathrm{T}(s'|s,a) > 0, a \in \mathcal{A}(s)\} = \emptyset$。

下面是我们进行动态规划函数的伪代码，尽管该算法看起来是递归的，但由于我们使用了一张动态变化的查询表来记录所有被计算过的 $V^*(s)$，因此，所有的三元组 (s,a,s') 都只需要遍历一次，就可以求得所有状态的最优值 $V^*(s)$。

```
for all state s do
    V*(s) ← NULL
    ▷ V*(s) is a global variable ◁

function Best-Value(s)
    if V*(s) is not NULL then
        return V*(s)
    else if s is terminal then
        V*(s) ← 0
        return V*(s)
    else
        for all a ∈ 𝒜(s) do
            Q(a|s) ← R(s,a) + Σ_{s'∈{T(s'|s,a)>0}} T(s'|s,a) Best-Value(s')
        V*(s) ← max_a Q(a|s)
        return V*(s)

for all state s do
    Best-Value(s)
```

2.3 用原 MDP 求解器求解衰减系数 $\gamma < 1$ 的 MDP 问题

原问题 我们要求解关于最优值函数 $V^*(s)$ 的贝尔曼方程为：

$$V^*(s) = \max_{a\in\mathcal{A}(s)} \sum_{s'} T(s'|s,a)\left[\mathrm{Reward}(s,a,s') + \gamma V^*(s')\right] \tag{1}$$

新问题 定义一个新的终点为 o，满足：

$$\mathrm{Reward}'(s,a,o) = 0, \quad V^*(o) = 0, \quad T'(o|s,a) = 1-\gamma$$

到达 o 不会获得任何奖励，在任何状态下采取任何动作都有 $1-\gamma$ 的概率到达新终点 o，并且到达后游戏就会结束；这也意味着在 o 处不存在任何最优策略。

对原问题(1)进行改写如下：

$$\begin{aligned}
V^*(s) &= \max_{a\in\mathcal{A}(s)} \sum_{s'} T(s'|s,a)\left[\mathrm{Reward}(s,a,s') + \gamma V^*(s')\right] \\
&= \max_{a\in\mathcal{A}(s)} \sum_{s'} \gamma T(s'|s,a)\left[\frac{\mathrm{Reward}(s,a,s')}{\gamma} + V^*(s')\right] \\
&= \max_{a\in\mathcal{A}(s)} \sum_{s'} \gamma T(s'|s,a)\left[\frac{\mathrm{Reward}(s,a,s')}{\gamma} + V^*(s')\right] + (1-\gamma)\times(0+0) \\
&= \max_{a\in\mathcal{A}(s)} \sum_{s'} \gamma T(s'|s,a)\left[\frac{\mathrm{Reward}(s,a,s')}{\gamma} + V^*(s')\right] + T'(o|s,a)(\mathrm{Reward}'(s,a,o) + V^*(o))
\end{aligned}$$

定义

$$T'(s'|s,a) = \gamma T(s'|s,a), \quad \mathrm{Reward}'(s,a,s') = \frac{\mathrm{Reward}(s,a,s')}{\gamma}$$

继续变形得到：

$$
\begin{aligned}
V^*(s) &= \max_{a\in\mathcal{A}(s)} \sum_{s'} T'(s'|s,a)\left[\text{Reward}'(s,a,s') + V^*(s')\right] + T'(o|s,a)(\text{Reward}'(s,a,o) + V^*(o)) \\
&= \max_{a\in\mathcal{A}(s)} \sum_{\tilde{s}\in s'\cup\{o\}} T'(\tilde{s}|s,a)\left[\text{Reward}'(s,a,\tilde{s}) + V^*(\tilde{s})\right] \qquad (2)
\end{aligned}
$$

注意到新问题(2)需要求解一个衰减系数 $\gamma' = 1$ 的贝尔曼公式，则可以使用只能求解 $\gamma' = 1$ 问题的 MDP 求解器进行求解。

3 问题 3: 将 Blackjack 实现为 MDP 并设计一套偷看率大于等于 10% 的牌

3.1 将 Blackjack 实现为 MDP

该部分为编程题,已在 `submission.py` 文件中的 `BlackjackMDP` 类下的 `succAndProbReward` 函数中完成。敬请参阅。

3.2 设计一套对至少 10% 的状态最佳策略是偷看的牌

我们设计的套牌，包含面值为 1，3，100 的牌各 100 张，已在 `submission.py` 文件中 `peekingMDP` 函数中完成。敬请参阅。

4 问题 4: 将强化学习应用于 Blackjack

4.1 实现通用的 Q-learning

已在 `submission.py` 文件中的 `QLearningAlgorithm` 类下的 `incorporateFeedback` 函数中实现。敬请参阅。

4.2 对比：Q-learninng 与 ValueIteration

我们在 `submission.py` 文件中的 `simulate_QL_over_MDP` 函数中添加了如下辅助代码：

Listing 1: 用于统计错误率结果的辅助代码 simulate_QL_over_MDP

```
def simulate_QL_over_MDP(mdp, featureExtractor):
    # NOTE: adding more code to this function is totally optional, but it will probably be useful
    # to you as you work to answer question 4b (a written question on this assignment).  We suggest
    # that you add a few lines of code here to run value iteration, simulate Q-learning on the MDP,
    # and then print some stats comparing the policies learned by these two approaches.
    # BEGIN_YOUR_CODE
    valueIteration = ValueIteration()
    valueIteration.solve(mdp)
    V_pi = valueIteration.pi # 由价值迭代所得到的最优策略
    QL = QLearningAlgorithm(mdp.actions, mdp.discount(), featureExtractor, explorationProb=0.2) # Q-learning
    util.simulate(mdp, QL, numTrials=30000, verbose=False) # 在simulate过程中会不断更新weights
    QL.explorationProb = 0 # 别忘了将Q-learning 算法的 explorationProb 设置为 0
    error, total = 0,  len(mdp.states)
    for state in mdp.states:
        if V_pi[state] != QL.getAction(state):
            error += 1
    print('最优动作错误的状态数:',error,' 总的状态数:',total)
    print('错误率为:  {:.3f}'. format(100 * error / total) + '%')
    # END_YOUR_CODE
```

考虑到 Q-learning 具有一定的随机性，因此我们进行了 5 次测试。我们的 5 轮测试结果如表10所示，其中我们认为 ValueIteration 给出的策略是最优策略，**每当 Q-learning 得到的动作与 ValueIteration 策略的最优动作不同时，我们记为 Q-learning 的一个错误。**

表 10: Q-learning 与 ValueIteration 对比试验

	smallMDP（总状态：38）		largeMDP（总状态：2950）	
实验序号	错误状态数	错误率	错误状态数	错误率
1	0	0	873	29.59%
2	1	2.63%	895	30.34%
3	1	2.63%	890	30.17%
4	2	5.26%	861	29.19%
5	1	2.63%	867	29.39%
平均	**1**	**2.63%**	**877.2**	**29.74%**

可以看出，在 smallMDP 问题上，Q-learning 策略与通过价值迭代学习的最优策略基本一致，错误率比较低。但是到了 largeMDP 问题上，Q-learning 策略的错误率显著升高，基本由 1/3 的策略与最优策略不一致。

我们认为可能有以下 3 个原因：

第一，30000 次试验不足以让 largeMDP 问题下的 Q-learning 算法收敛；

第二，即便收敛，由于我们在 Q-learning 采用了 ϵ-greedy 算法默认提供的探索率（explorationProb=0.2），该探索率相对较低，在 largeMDP 问题下，搜索空间变得非常巨大，这或

许会导致 Q-learning 对于某些状态没有进行探索，陷入局部最优解；

第三，在本题中使用的特征提取器只提取当前的 `state` 和 `action` 作为特征，**泛化能力较差**。我们推测，第三个原因可能是 Q-learning 在 largeMDP 问题上表现不良的主要原因。

4.3 构造高泛化能力的特征提取器

我们已经在 `submission.py` 文件中的 `blackjackFeatureExtractor` 函数中实现了具有较好领域泛化能力的特征提取器，敬请参阅。在使用新的特征提取器后，Q-learning 与 ValueIteration 对比试验结果如下

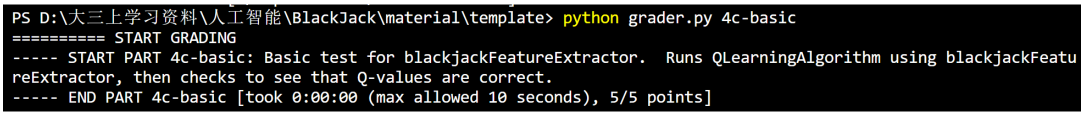


图 1: 4c 测试结果

这说明在使用具有较强泛化能力的特征提取器后，Q-learning 的表现能够逼近 ValueIteration。

4.4 将最优策略应用于规则改变后的 MDP 问题

我们在 `submission.py` 文件的 `compare_changed_MDP` 函数中补充了如下辅助内容：

Listing 2: 4d 问题的辅助函数

```python
def compare_changed_MDP(original_mdp, modified_mdp, featureExtractor):
    # NOTE: as in 4b above, adding more code to this function is completely optional, but we've added
    # this partial function here to help you figure out the answer to 4d (a written question).
    # Consider adding some code here to simulate two different policies over the modified MDP
    # and compare the rewards generated by each.
    # BEGIN_YOUR_CODE
    from util import ValueIteration
    vIteration = ValueIteration()
    vIteration.solve(original_mdp)

    # 通过FixedRLAlgorithm 实例调用，基于旧的策略
    RL_old = util.FixedRLAlgorithm(vIteration.pi)
    rewards = util.simulate(original_mdp,RL_old, numTrials=10000)
    print('由旧的策略得到的平均回报为：', sum(rewards) /  len(rewards))

    # 使用 blackjackFeatureExtractor 和默认探索概率直接在 newThresholdMDP 上模拟 Q-learning
    RL_QL = QLearningAlgorithm(original_mdp.actions, original_mdp.discount(), featureExtractor)
    rewards = util.simulate(modified_mdp,RL_QL, numTrials=10000)
    print('由基于新的特征提取器的Q-learning得到的平均回报为：', sum(rewards) /  len(rewards))
    # END_YOUR_CODE
```

在该函数中，我们先基于值迭代，得到了对于旧问题的最优策略，然后我们将旧策略用到新问题上，进行 10000 次尝试，并将 10000 次尝试的回报取平均值，作为本轮试验的回报。类

似的，我们使用在上一题中完成的 `blackjackFeatureExtractor` 和默认探索概率（0.2）直接在 newThresholdMDP 上进行 Q-learning，同样进行 10000 次尝试，并把 10000 次尝试的回报取平均值，作为本轮试验的回报。

由于马尔可夫过程具有随机性，导致回报也有随机性。为减小随机性的影响，我们进行了五轮试验，结果如表11所示：

表 11: 原策略与 Q-learning 在 newThresholdMDP 的回报对比

实验序号	原策略平均回报	Q-learning 平均回报
1	6.8386	9.5464
2	6.8442	9.5215
3	6.8394	9.3860
4	6.8215	9.4801
5	6.8067	9.5144
平均	**6.8301**	**9.4897**

可以看出，使用 Q-learning 的平均回报，明显高于使用旧策略的平均回报（大约多了 50%）。我们认为原因有以下两个：

第一，根据上一题的试验与讨论，使用 `blackjackFeatureExtractor` 作为特征提取器的 Q-learning 具有很好的领域泛化的能力，表现几乎和价值迭代一致，因此由 Q-learning 给出的策略会逐渐逼近 newThresholdMDP 的最优策略，也即由 Q-learning 给出的回报接近 newThresholdMDP 的最优的回报。

第二，在规则改变后，最优策略会发生变化，但旧的策略是基于旧的 originalMDP 问题，显然不是最优策略，因此得到的平均回报也会较少。