强化学习基础篇（十六）蒙特卡洛预测算法在21点游戏的应用

本节将介绍Monte Carlo prediction算法在Blackjack游戏中的进行预测的过程。主要基于一个最简单的策略进行评估，即"超过18点就不在要牌，低于18点就继续要牌"。我将使用两种类型的算法进行评估，一个是首次访问型蒙特卡洛预测算法（First-visit MC prediction），另一个是每次访问型蒙特卡洛预测算法（Every-visit MC prediction）。

1、 首次访问型MC预测算法

回顾一下前面介绍的首次访问型MC预测算法。

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$

2、21点游戏

21点游戏使用一副或多副标准的52张纸牌，每张牌都规定一个点值。2~10的牌其点值按面值计算。J、Q和K都算作10点，A可算作1点，也可算作11点。玩家的目标是所抽牌的总点数比庄家的牌更接近21点，但不超过21点。

首次发牌每人2张牌。庄家以顺时针方向向众玩家派发一张暗牌（即不被揭开的牌），随后向自己派发一张暗牌；接着庄家会以顺时针方向向众玩家派发一张明牌（即被揭开的牌），之后向自己也派发一张明牌。当众人手上各拥一张暗牌和一张明牌时，庄家就以顺时针方向逐位询问玩家是否再要牌（以明牌方式派发）。在要牌的过程中。如果互家所有的牌加起来超过21点，玩家就输了（Bust），游戏介绍，该玩家的注码归庄家。

如果玩家无Bust，庄家询问完所有玩家之后，就必须揭开自己上上的暗牌。若庄家总点数少于17点，就必须继续要牌；如果庄家Bust，便向没有Bust的玩家，赔出该玩家所投的同等注码。如果庄家无Bust且大于等于17点，那么庄家与玩家比较点数决胜负，大的为赢。点数相同，则为平手。

在该21点游戏例子中，收集经验轨迹时，首先需要确认该游戏基于基策路π下，进行经验数据收集。

为了便于理解，我们使用一个简单的策略，当玩家手上的牌超过18点时，返回0，表示不再要牌；当点数少于18点时，继续要牌，并返回1。

```python
def simple_policy(state):
    """
    定义个简单的策略，当玩家手上的牌超过18点时，返回0，
    表示不再要牌(hold)；当点数少于18点时，继续要牌（hit），并返回1。
    """
    player_score, _, _ = state
    return 0 if player_score >= 18 else 1
```

旅游戏的状态是玩家的点数（Player），庄家的点数（Dealer）和是否有Blackjack(Ace)。具体到代码中，player为玩家点数，dealer为庄家点数，ace为True时表明牌A算作11点，

对于21点游戏，简化版的玩家动作只有两种：一种是拿牌，另一种是停牌。

- 拿牌（HIT）：如果玩家拿牌，表示玩家希望再拿一张或多张牌，使总点数更接近21点。如果拿牌后玩家的总点数超过21点，玩家就会Bust。
- 停牌（STAND）：如果玩家停牌，表示玩家选择不再抽牌并希望当前总点数能够打败庄家。

3.First-visit MC prediction源码清单

```python
# coding: utf-8

import numpy as np
import gym
import sys
from collections import defaultdict
from Plot3D import plot_3D

def mc_firstvisit_prediction(policy, env, num_episodes,
                             episode_endtime= 10, discount = 1.0):
    """
    该函数主要实现首次访问蒙特卡洛预测算法
    """

    r_sum = defaultdict(float)
    r_count = defaultdict(float)
    r_V = defaultdict(float)

    # 按照设定的num_episodes数量进行对应迭代
    for each_episode in range(num_episodes):
        # 打印迭代进展
        print("Episode {}/{}".format(each_episode,num_episodes),end = "\r")
        sys.stdout.flush()

        # 将episode初始化为列表
        episode = []
        # 重置环境
        state = env.reset()

        # 按照输入的策略，采集episode
        for _ in range(episode_endtime):
            # 获取当前状态是预定策略将会采取的动作
            action = policy(state)
            # 与环境交互，获取下一个状态，奖励，以及结束标志位
            next_state, reward, done, info = env.step(action)
            # 将(s,a,r)对插入episode中
            episode.append((state, action, reward))
```

```
            if done:
                break
            state = next_state


        # 计算首次访问蒙特卡洛算法的值
        for visit_pos, data in enumerate(episode):
            # 遍历episode过程中，state_visit为当前遍历的访问状态
            state_visit = data[0]
            # x[2]为reward，这里为对首次访问后的所有奖励都做带discount的累加
            G = sum([x[2] * np.power(discount, i) for i, x in
enumerate(episode[visit_pos:])])

            # 计算累积平均奖励
            r_sum[state_visit] += G
            r_count[state_visit] += 1.0
            r_V[state_visit] = r_sum[state_visit] / r_count[state_visit]

    return r_V

def simple_policy(state):
    """
    定义个简单的策略，当玩家手上的牌超过18点时，返回0，
    表示不再要牌(hold)；当点数少于18点时，继续要牌（hit），并返回1。
    """
    player_score, _, _ = state
    return 0 if player_score >= 18 else 1

def process_data_for_Blackjackproblem(V,ace=True):
    """
    为Blackjack问题进行3D画图处理
    """
    min_x = min(k[0] for k in V.keys())
    max_x = max(k[0] for k in V.keys())
    min_y = min(k[1] for k in V.keys())
    max_y = max(k[1] for k in V.keys())

    x_range = np.arange(min_x, max_x + 1)
    y_range = np.arange(min_y, max_y + 1)
    X, Y = np.meshgrid(x_range, y_range)

    if ace:
        Z = np.apply_along_axis(lambda _ : V[(_[0], _[1], True)], 2,
np.dstack([X,Y]))
    else:
        Z = np.apply_along_axis(lambda _ : V[(_[0], _[1], False)], 2,
np.dstack([X,Y]))

    return X, Y, Z

if __name__ == "__main__":
    # 调用gym的Blackjack-v0环境
    env = gym.make("Blackjack-v0")
    # 允许100万次的首次访问蒙特卡洛预测算法，并返回值函数
    v1= mc_firstvisit_prediction(simple_policy, env, num_episodes=1000000)
    print(v1)
    # 进行3D画图数据处理
    X, Y, Z = process_data_for_Blackjackproblem(v1, ace=True)
```

```
    fig = plot_3D(X, Y, Z, xlabel="Player sum", ylabel="Dealer sum",
zlabel="Value", title="Usable Ace")
    fig.show()
    fig.savefig("./log/Usable_Ace.jpg")
    X, Y, Z = process_data_for_Blackjackproblem(v1, ace= False)
    fig = plot_3D(X, Y, Z, xlabel="Player sum", ylabel="Dealer sum",
zlabel="Value", title="No Usable Ace")
    fig.show()
    fig.savefig("./log/No_Usable_Ace.jpg")

```

- mc_firstvisit_prediction（）方法定义了4个输入：

  policy：定义的策略

  env：环境

  num_episodes：采样的幕的数量

  episode_endtime= 10：设定个幕的最大数量

  discount ： 折扣因子

- 首先使用defaultdict定义了过程中使用的字典，defaultdict相比dict，当字典里的key不存在但被查找时，返回的不是keyError而是一个空默认值。
- gym返回的状态有一个三元组组成，分别表示玩家当前手上牌的总数，庄家的牌点数（1为ace），以及玩家是否有ace的标志。例如$(14, 5, False)$表示玩家当前手上牌的总数为14，庄家明牌点数为5，玩家当前无ace。
- 以下代码通过与环境交互可以获得真实的episode。

```
        for _ in range(episode_endtime):
            # 获取当前状态是预定策略将会采取的动作
            action = policy(state)
            # 与环境交互，获取下一个状态，奖励，以及结束标志位
            next_state, reward, done, info = env.step(action)
            # 将(s,a,r)对插入episode中
            episode.append((state, action, reward))
            if done:
                break
            state = next_state
```

结果：

```
[((4, 6, False), 1, 0.0), ((7, 6, False), 1, 0.0), ((16, 6, False), 1,
0.0), ((17, 6, False), 1, -1.0)]
```

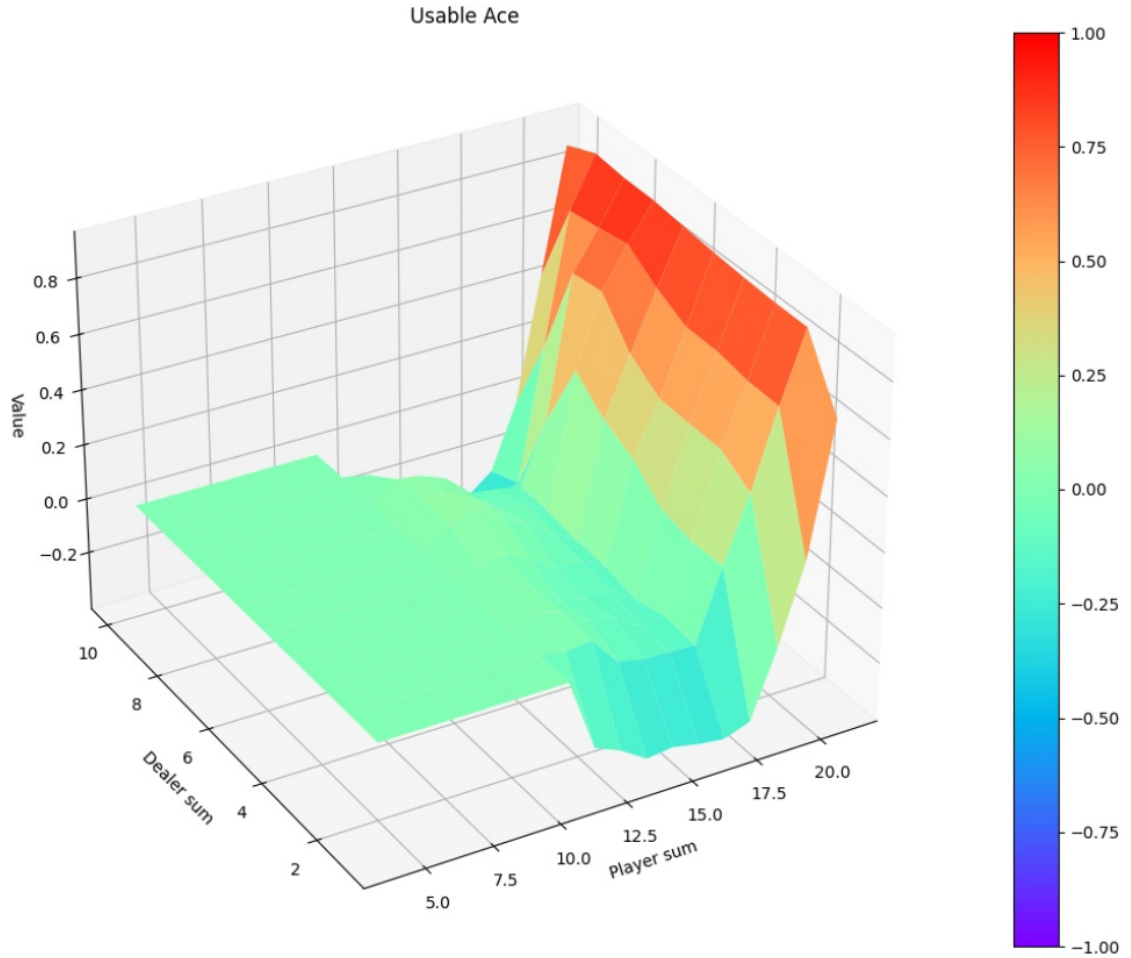
结果为一幕从开始到结束的关于(state, action,reward)的三元组。

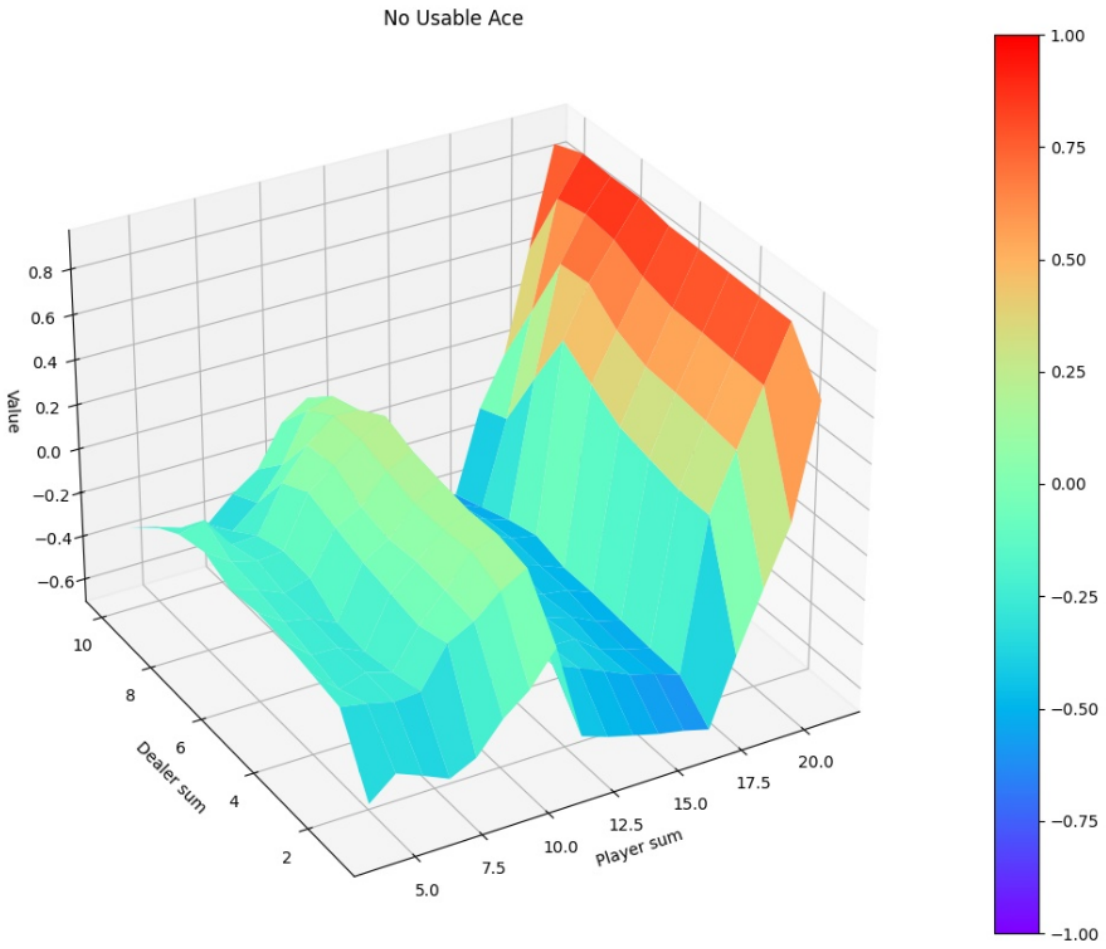
- 允许100万幕的首次访问蒙特卡洛预测的价值函数如下所示,其中包含了每个状态的价值函数。

```
{
    (19, 2, False): 0.3803322395406071,
    (14, 8, False): -0.37651049395802416,
    (20, 8, False): 0.7971809523809524,
    (12, 6, False): -0.3004375863657301,
    (18, 6, False): 0.28861230889847117,
    (18, 10, True): -0.252530199151159,
.............................
    (12, 5, True): 0.013071895424836602,
    (12, 3, True): 0.03160270880361174
}
```

4. First-visit MC prediction测试结果

下面两张图为我们对简单策略“超过18点就不在要牌，低于18点就继续要牌”下，对应的状态值在有可用Ace与无可用Ace下的价值在三维空间的分布情况。颜色越深，状态值越高。





5.每次访问型蒙特卡洛预测算法（Every-visit MC prediction）源码清单

```
# coding: utf-8

import numpy as np
import gym
import sys
from collections import defaultdict
from Plot3D import plot_3D

def mc_everyvisit_prediction(policy, env, num_episodes,
                             episode_endtime= 10, discount = 1.0):
    """
    该函数主要实现每次访问蒙特卡洛预测算法
    """
    r_sum = defaultdict(float)
    r_count = defaultdict(float)
    r_V = defaultdict(float)
    # 按照设定的num_episodes数量进行对应迭代
    for each_episode in range(num_episodes):
        # 打印迭代进展
        print("Episode {}/{}".format(each_episode,num_episodes),end = "\r")
        sys.stdout.flush()

```

```
        # 将episode初始化为列表
        episode = []
        state = env.reset()

        # 按照输入的策略，采集episode
        for _ in range(episode_endtime):
            action = policy(state)
            # 与环境交互，获取下一个状态，奖励，以及结束标志位
            next_state, reward, done, info = env.step(action)
            # 将(s,a,r)对插入episode中
            episode.append((state, action, reward))
            if done:
                break
            state = next_state


        # 计算首次访问蒙特卡洛算法的价值
        for visit_pos, data in enumerate(episode):
            state_visit = data[0]
            # x[2]为reward, 这里对所有奖励都做带discount的累加
            G = sum([x[2] * np.power(discount, i) for i, x in
enumerate(episode)])

            # 计算累积平均奖励
            r_sum[state_visit] += G
            r_count[state_visit] += 1.0
            r_V[state_visit] = r_sum[state_visit] / r_count[state_visit]

    return r_V

def simple_policy(state):
    """
    定义个简单的策略，当玩家手上的牌超过18点时，返回0，
    表示不再要牌(hold)；当点数少于18点时，继续要牌（hit），并返回1。
    """
    player_score, _, _ = state
    return 0 if player_score >= 18 else 1

def process_data_for_Blackjackproblem(V,ace=True):
    """
    为Blackjack问题进行3D画图处理
    """
    min_x = min(k[0] for k in V.keys())
    max_x = max(k[0] for k in V.keys())
    min_y = min(k[1] for k in V.keys())
    max_y = max(k[1] for k in V.keys())

    x_range = np.arange(min_x, max_x + 1)
    y_range = np.arange(min_y, max_y + 1)
    X, Y = np.meshgrid(x_range, y_range)

    if ace:
        Z = np.apply_along_axis(lambda _ : V[(_[0], _[1], True)], 2,
np.dstack([X,Y]))
    else:
        Z = np.apply_along_axis(lambda _ : V[(_[0], _[1], False)], 2,
np.dstack([X,Y]))

```

```
    return X, Y, Z

if __name__ == "__main__":
    # 调用gym的Blackjack-v0环境
    env = gym.make("Blackjack-v0")
    # 允许100万次的首次访问蒙特卡洛预测算法，并返回值函数
    v1= mc_everyvisit_prediction(simple_policy, env, num_episodes=1000000)
    print(v1)
    # 进行3D画图数据处理
    X, Y, Z = process_data_for_Blackjackproblem(v1, ace=True)
    fig = plot_3D(X, Y, Z, xlabel="Player sum", ylabel="Dealer sum",
zlabel="Value", title="Usable Ace")
    fig.savefig("./log/EveryVisit_Usable_Ace_1M.jpg")
    X, Y, Z = process_data_for_Blackjackproblem(v1, ace= False)
    fig = plot_3D(X, Y, Z, xlabel="Player sum", ylabel="Dealer sum",
zlabel="Value", title="No Usable Ace")
    fig.savefig("./log/EveryVisit_No_Usable_Ace_1M.jpg")

```

- 首次访问型蒙特卡洛预测算法（First-visit MC prediction）与每次访问型蒙特卡洛预测算法（Every-visit MC prediction）几乎完全相同，唯一的区别在于计算未来折扣奖励方式的不同。

  在每次访问蒙特卡洛预测算法中，每采集完一条经验轨迹后，同样首次访问型蒙特卡洛预测算法的方式对未来折扣累积奖励进行计算，作为状态值的期望。其中，算法使用到的参数都完全相同，r_sum表示该条经验轨迹的总回报，r_count表示该条经验轨迹的统计次数，r_V表示总体的状态值。

  每次访同蒙特卡洛预测算法的核心在于：无论状态s出现多少次，每一次的些励返回值都被纳入平均未来折扣累积奖励的计算：

  ```
  G = sum([x[2] * np.power(discount, i) for i, x in enumerate(episode)])
  ```

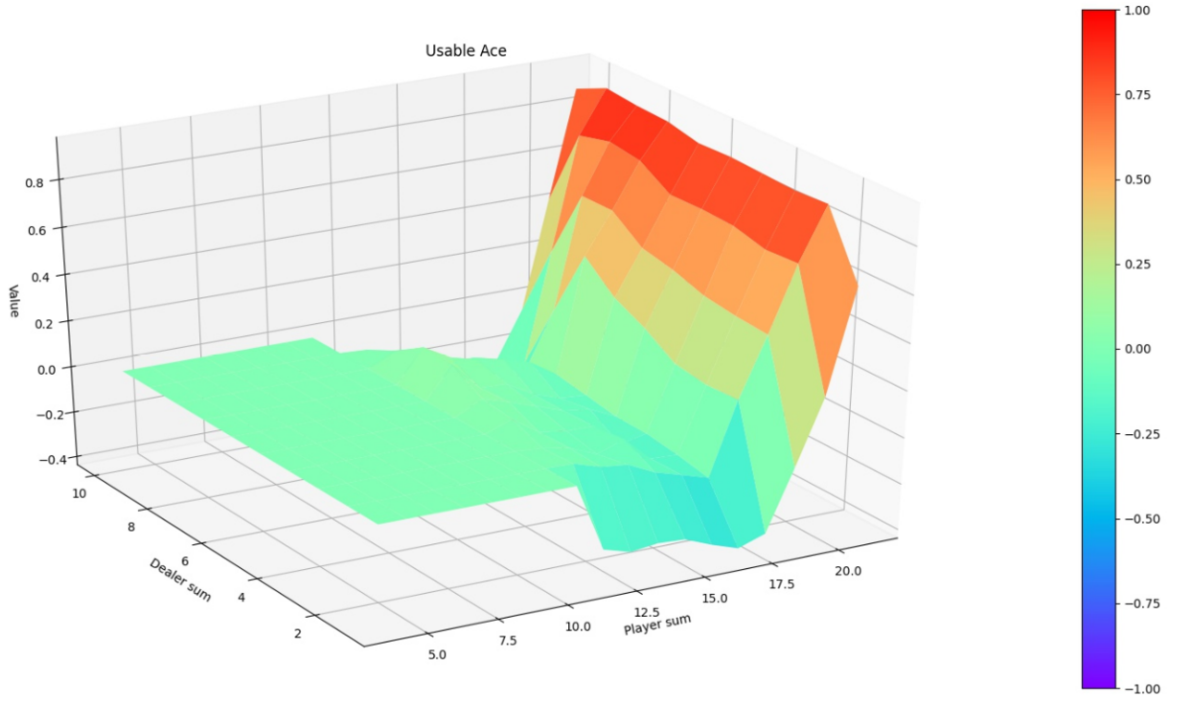
  这里与首次奖励的计算方式有着细微的表述差异：

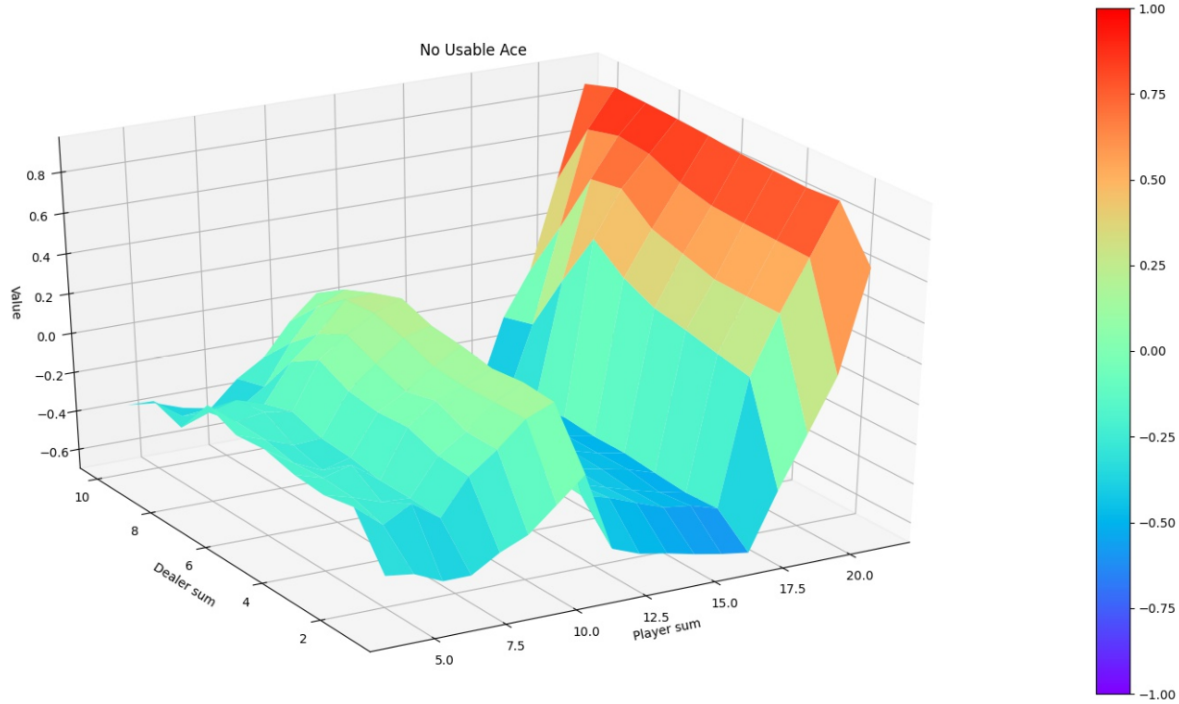
  ```
  G = sum([x[2] * np.power(discount, i) for i, x in
  enumerate(episode[visit_pos:])])
  ```

5.Every-visit MC prediction测试结果

下面两张图为我们对简单策略“超过18点就不在要牌，低于18点就继续要牌”下，对应的状态值在有可用Ace与无可用Ace下的价值在三维空间的分布情况。颜色越深，状态值越高。

其实看起来测试结果与首次访问MC预测算法测试结果差异不大。


Usable Ace
Value
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
Dealer sum
10
8
6
4
2
Player sum
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


No Usable Ace
Value
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
Dealer sum
10
8
6
4
2
Player sum
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00