H.W/ Use first-visit monte Carlo policy evaluation To calculate blackjack value function

A reinforcement learning algorithm called "First-Visit Monte Carlo Policy Evaluation" is used to determine the value function of a given policy in a Markov Decision Process (MDP). Let's dissect this algorithm's main elements and steps:

* Monte Carlo Methods:

A class of algorithms known as Monte Carlo methods uses random sampling to produce numerical results. By averaging over several random samples (episodes), Monte Carlo methods are frequently used in the context of reinforcement learning to estimate the expected return of a policy.

Policy Evaluation:

Finding a policy's value function is the process of policy evaluation. The expected cumulative reward that an agent can anticipate receiving when adhering to a particular policy is represented by the value function.

Steps of First-Visit Monte Carlo Policy Evaluation:

* Generate Episodes

Using the policy that is going to be assessed, the algorithm begins by creating episodes. An episode is a series of conditions, deeds, and rewards that start at one point and end when the episode ends.

* First-Visit Check:

The algorithm determines if a state is encountered for the first time in an episode for each state. The state is added to a list of visited states if this is the first visit.

* Accumulate Returns:

The algorithm keeps track of the returns, or the total of the rewards, from each state until the end of the episode for each initial visit.

* Update Value Function:

The cumulative returns are used to update the value function for each state that is visited. Usually, an incremental update formula that accounts for the number of visits to the state is used for this.

* Repeat

After a predetermined number of episodes, steps 1-4 are repeated. The objective of averaging across several episodes is to derive more precise estimates of the state values.

* Output Estimated Value Function:

The estimated value function, which associates each state with its estimated cumulative reward, is the algorithm's final output.

blackjack:

It's a skill-and luck-based card game. The objective is to win by getting the score as close to 21 as possible without going over that amount while competing against the casino. The objective is to get the sum of each player's card values as close to 21 as possible. Players are identified by their cards.

A standard deck of cards is used to play blackjack, and each card has a value assigned to it. The numerical values of cards 2 through 10 are carried by the heads (king, queen, and month), while the value of the ace is dependent on who serves and can range from 1 to 11. Finding the card total that is closest to 21 without going over is the object of the game.

After each player is dealt two cards at first, they can choose to stand when they believe their hand is nearly 21 ("surrender") or call for more cards ("hit"). It is also possible to divide two identical cards into different hands or to double the wager.

A player is eliminated if their card total is greater than 21 (a "blow"). A player wins the trick if he gets cards that add up to 21 or if his total is closer to 21 than the casino's.

Blackjack provides players with good strategic options because they can decide how to increase their chances of winning by considering both their own cards and the casino's up cards.

import numpy as np

from collections import defaultdict

def generate\_episode(policy):

    # With the provided policy, create an episode.

    episode = []

    # Example

    # Assume that the player and dealer are playing a straightforward game where the winner is the sum of all the players' cards.

    player\_sum = 0

    while player\_sum < 20:

        card = np.random.randint(1, 11)

        player\_sum += card

        episode.append((player\_sum, "hit", 0))

# The winner is the total of all players.

    reward = 1 if player\_sum <= 21 else 0

    episode[-1] = (episode[-1][0], "stand", reward)

    return episode

def first\_visit\_mc\_policy\_evaluation(policy, num\_episodes):

    # Set the state value function to initial values.

    V = defaultdict(float)

    # First-time visitor counter

    N = defaultdict(int)

    for episode\_num in range(num\_episodes):

        episode = generate\_episode(policy)

# Before unpacking, make sure the episode is not empty.

        if episode:

             # Extract states, actions, and rewards from the episode

            states, \_, rewards = zip(\*episode)

            # Update state values

            visited\_states = set()

            for t, state in enumerate(states):

                if state not in visited\_states:

                    visited\_states.add(state)

                    G = sum(rewards[t:])

                    N[state] += 1

                    V[state] += (G - V[state]) / N[state]

    return V

def simple\_policy(player\_sum):

    return "stand" if player\_sum >= 20 else "hit"

def print\_value\_function(value\_function):

    for state, value in value\_function.items():

        print(f"State: {state}, Estimated Value: {value}")

def main():

    num\_episodes = 500

    estimated\_value\_function = first\_visit\_mc\_policy\_evaluation(simple\_policy, num\_episodes)

    # Print the estimated value function

    print\_value\_function(estimated\_value\_function)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

OUTPUT

State: 1, Estimated Value: 0.3965517241379311

State: 10, Estimated Value: 0.32539682539682535

State: 18, Estimated Value: 0.23809523809523822

State: 28, Estimated Value: 0.0

State: 7, Estimated Value: 0.36363636363636376

State: 16, Estimated Value: 0.3076923076923077

State: 25, Estimated Value: 0.0

State: 2, Estimated Value: 0.4716981132075471

State: 9, Estimated Value: 0.3009708737864078

State: 12, Estimated Value: 0.4000000000000001

State: 17, Estimated Value: 0.24418604651162795

State: 20, Estimated Value: 1.0

State: 14, Estimated Value: 0.2840909090909091

State: 21, Estimated Value: 1.0

State: 26, Estimated Value: 0.0

State: 4, Estimated Value: 0.35384615384615387

State: 22, Estimated Value: 0.0

State: 23, Estimated Value: 0.0

State: 13, Estimated Value: 0.475

State: 19, Estimated Value: 0.1222222222222222

State: 29, Estimated Value: 0.0

State: 11, Estimated Value: 0.426829268292683

State: 8, Estimated Value: 0.39534883720930236

State: 6, Estimated Value: 0.36000000000000004

State: 3, Estimated Value: 0.4814814814814815

State: 5, Estimated Value: 0.32857142857142874

State: 27, Estimated Value: 0.0

State: 24, Estimated Value: 0.0

State: 15, Estimated Value: 0.3026315789473685

