МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__4__
по дисциплине «Методы машинного обучения»

Тема: «Реализация алгоритма Policy Iteration.»

ИСПОЛНИТЕЛЬ: Кузьмин Р.А.
ФИО
группа ИУ5-25М
подпись
" __"_ _____2024 г.

ПРЕПОДАВАТЕЛЬ: Гапанюк Ю.Е.
ФИО
подпись
"__"_____2024 г.

Москва - 2024

Задание

1. На основе рассмотренного на лекции примера реализуйте алгоритм Policy Iteration для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).
2. Сформировать отчет и разместить его в своем репозитории на github.

Выполнение

Для реализации была выбрана среда Cliffwalking-v0 из библиотеки Gym.

Cliffwalking предполагает пересечение мира с сеткой от начала до цели, избегая при этом падения с обрыва. Игра начинается с того, что игрок оказывается в локации [3, 0] мира с сеткой 4x12, а цель находится в точке [3, 11]. Если игрок достигает цели, эпизод заканчивается. Обрыв проходит вдоль [3, 1..10]. Если игрок перемещается к месту, где находится обрыв, он возвращается в исходное положение. Игрок делает ходы, пока не достигнет цели.

По документации: 48 состояний – (карта)

4 действия – по 4ем направлениям движения

Существует 3 x 12 + 1 возможных состояния. Игрок не может находиться ни на скале, ни у цели, поскольку последнее приводит к окончанию эпизода. Остаются все позиции в первых 3 рядах плюс нижняя левая ячейка.

Observation - это значение, представляющее текущую позицию игрока в виде current_row * nrows + current_col (и строка, и столбец начинаются с 0).

Код программы:

```
import gym
import numpy as np
import matplotlib.pyplot as plt
from pprint import pprint
import pandas as pd
from gym.envs.toy_text.cliffwalking import CliffWalkingEnv


def print_full(x):
    pd.set_option('display.max_rows', len(x))
    print(x)
    pd.reset_option('display.max_rows')
```

```
class PolicyIterationAgent:
    '''
    Класс, эмулирующий работу агента
```

```
    '''
    def __init__(self, env):
        self.env = env
        # Пространство состояний
        self.observation_dim = 48
        # Массив действий в соответствии с документацией
        # https://www.gymlibrary.dev/environments/toy_text/cliff_walking/
        self.actions_variants = np.array([0,1,2,3])
        # Задание стратегии (политики)
        self.policy_probs = np.full((self.observation_dim, len(self.actions_variants)), 0.25)
        # Начальные значения для v(s)
        self.state_values = np.zeros(shape=(self.observation_dim))
        # Начальные значения параметров
        self.maxNumberOfIterations = 1000
        self.theta=1e-6
        self.gamma=0.99
```

```
    def print_policy(self):
        '''
        Вывод матриц стратегии
        '''
        print('Стратегия:')
        pprint(self.policy_probs)
```

```
    def policy_evaluation(self):
        '''
        Оценивание стратегии
        '''
        # Предыдущее значение функции ценности
        valueFunctionVector = self.state_values
        for iterations in range(self.maxNumberOfIterations):
            # Новое значение функции ценности
            valueFunctionVectorNextIteration=np.zeros(shape=(self.observation_dim))
            # Цикл по состояниям
            for state in range(self.observation_dim):
                # Вероятности действий
                action_probabilities = self.policy_probs[state]
                # Цикл по действиям
                outerSum=0
                for action, prob in enumerate(action_probabilities):
                    innerSum=0
                    # Цикл по вероятностям действий
                    for probability, next_state, reward, isTerminalState in
self.env.P[state][action]:
                        innerSum=innerSum+probability*(reward+self.gamma*self.state_values[next_s
tate])
                    outerSum=outerSum+self.policy_probs[state][action]*innerSum
                valueFunctionVectorNextIteration[state]=outerSum
            if(np.max(np.abs(valueFunctionVectorNextIteration-valueFunctionVector))<self.theta):
                # Проверка сходимости алгоритма
                valueFunctionVector=valueFunctionVectorNextIteration
                break
            valueFunctionVector=valueFunctionVectorNextIteration
        return valueFunctionVector
```

```
    def policy_improvement(self):
        '''
        Улучшение стратегии
```

```
        '''
        qvaluesMatrix=np.zeros((self.observation_dim, len(self.actions_variants)))
        improvedPolicy=np.zeros((self.observation_dim, len(self.actions_variants)))
        # Цикл по состояниям
        for state in range(self.observation_dim):
            for action in range(len(self.actions_variants)):
                for probability, next_state, reward, isTerminalState in self.env.P[state][action]:
                    qvaluesMatrix[state,action]=qvaluesMatrix[state,action]+probability*(reward+s
elf.gamma*self.state_values[next_state])
```

```
            # Находим лучшие индексы
            bestActionIndex=np.where(qvaluesMatrix[state,:]==np.max(qvaluesMatrix[state,:]))
            # Обновление стратегии
            improvedPolicy[state,bestActionIndex]=1/np.size(bestActionIndex)
        return improvedPolicy
```

```
    def policy_iteration(self, cnt):
        '''
        Основная реализация алгоритма
        '''
        policy_stable = False
        for i in range(1, cnt+1):
            self.state_values = self.policy_evaluation()
            self.policy_probs = self.policy_improvement()
        print(f'Алгоритм выполнился за {i} шагов.')
```

```
def play_agent(agent):
    env2 = gym.make('CliffWalking-v0', render_mode='human')
    state = env2.reset()[0]
    done = False
    while not done:
        p = agent.policy_probs[state]
        if isinstance(p, np.ndarray):
            action = np.random.choice(len(agent.actions_variants), p=p)
        else:
            action = p
        next_state, reward, terminated, truncated, _ = env2.step(action)
        env2.render()
        state = next_state
        if terminated or truncated:
            done = True
```

```
def main():
    # Создание среды
    env = gym.make('CliffWalking-v0')
    env.reset()
    # Обучение агента
    agent = PolicyIterationAgent(env)
    agent.print_policy()
    agent.policy_iteration(1000)
    agent.print_policy()
    # Проигрывание сцены для обученного агента
    play_agent(agent)
```

```
if __name__ == '__main__':
    main()
```

Результаты выполнения:

```
Стратегия:
array([[0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
```

Рис. 1 – начальная стратегия.

Фрагмент вывода стратегии:

```
[[0. , 0.5 , 0.5 , 0. ],
[0.33333333, 0.33333333, 0.33333333, 0. ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.33333333, 0. , 0.33333333, 0.33333333],
[0. , 0. , 0.5 , 0.5 ],
[0. , 0.33333333, 0.33333333, 0.33333333],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0. , 0.33333333, 0.33333333, 0.33333333],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.33333333, 0.33333333, 0. , 0.33333333],
[0.25 , 0.25 , 0.25 , 0.25 ],
[0.33333333, 0. , 0.33333333, 0.33333333],
[0.5 , 0. , 0. , 0.5 ], [1. , 0. , 0. , 0. ],
```

```
[1. , 0. , 0. , 0. ], [1. , 0. , 0. , 0. ],
[1. , 0. , 0. , 0. ], [1. , 0. , 0. , 0. ],
[1. , 0. , 0. , 0. ], [1. , 0. , 0. , 0. ],
[1. , 0. , 0. , 0. ], [0.5 , 0.5 , 0. , 0. ],
[0.33333333, 0.33333333, 0.33333333, 0. ]]
```

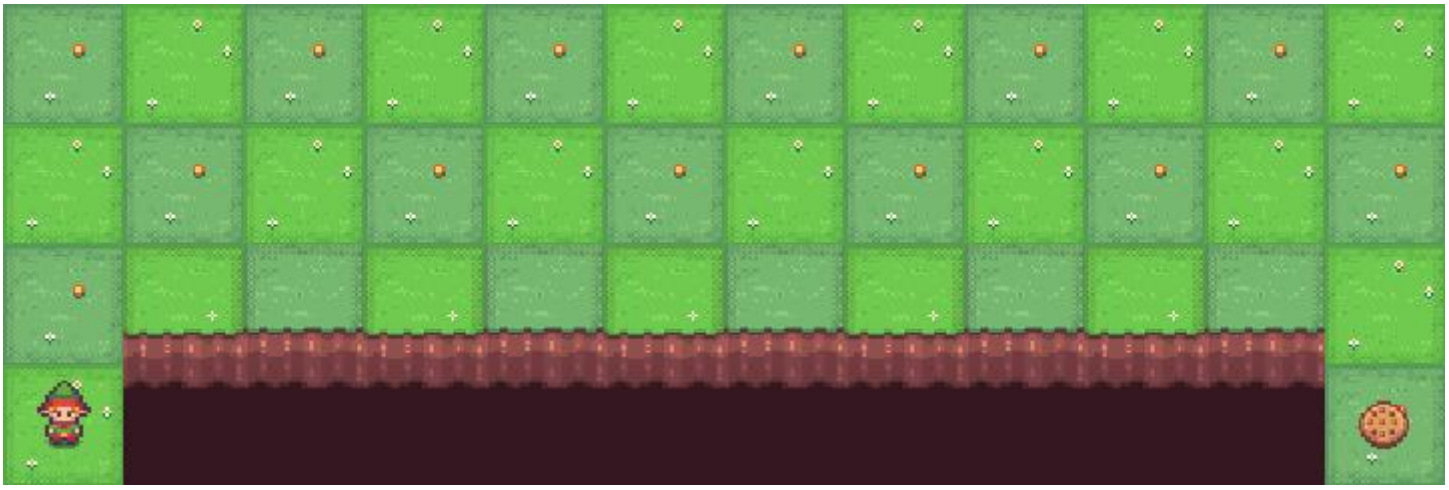

Рис. 2 – работа программы.

Вывод:

Методика Policy Iteration позволяет, имея матрицу состояний и вероятностей действий, итеративно улучшать стратегию переходов между состояниями. В данной ЛР улучшение достигается за счёт штрафа за лишние переходы и штрафов за падение с обрыва.

За каждый шаг начисляется штраф в -1 балл, если игрок не упал с обрыва, иначе начисляется штраф в размере -100 баллов. За 1000 итераций качество обучения становится достаточно хорошим, но, теоретически, достаточно и 50.