

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ
им. Н.Э. Баумана

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

ОТЧЕТ

Лабораторная работа №__5__
по дисциплине «Методы машинного обучения»

Тема: «Реализация алгоритма Policy Iteration.»

ИСПОЛНИТЕЛЬ:

Коротков Н.К.

ФИО

группа

ИУ5-23М

подпись

"__"__2024 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

ФИО

подпись

"__"__2024 г.

Москва - 2024

Задание

1. На основе рассмотренного на лекции примера реализуйте следующие алгоритмы:

- a. SARSA
- b. Q-обучение
- c. Двойное Q-обучение

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 Taxi-v3 из библиотеки Gym.

По документации: 500 состояний – 5*5 карта, 4 возможных локации точки выхода, 5 состояний пассажира (4 выхода и в такси).

6 действий – 4 движения и взять/высадить пассажира.

Из 500 состояний в рамках 1 итерации достижимо 400 – исключаются состояния где пассажир там же, где и здание.

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.taxi import TaxiEnv

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 = 500
```

```

# Массив действий в соответствии с документацией
self.actions_variants = np.array([0,1,2,3,4,5])
# Задание стратегии (политики)
self.policy_probs = np.full((self.observation_dim,
len(self.actions_variants)), 0.16666666)
# 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
    '''

    if self.policy_probs[0][0] != 0.16666666:
        #np.set_printoptions(threshold=np.inf)
        x = TaxiEnv()
        pos = {0:'R', 1:'G',2:'Y', 3:'B', 4:'T'}
        print(''
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+-----+
        ''')
        print('состояние: x,y,пассажир,назначение')
        print('Стратегия:')
        for i in range(len(self.policy_probs)):
            t_x,t_y,passeng,dest = x.decode(i)
            print((t_x,t_y,pos[passeng],pos[dest]), self.policy_probs[i])
            #np.set_printoptions(threshold=False)
        else:
            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_state])
                    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+self.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 выполнен за {i} шагов.')

def play_agent(agent):
    env2 = gym.make('Taxi-v3', 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('Taxi-v3')
    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()

```

Вывод программы модифицирован для кодировки состояний: см. рис. 2.

Результаты выполнения:

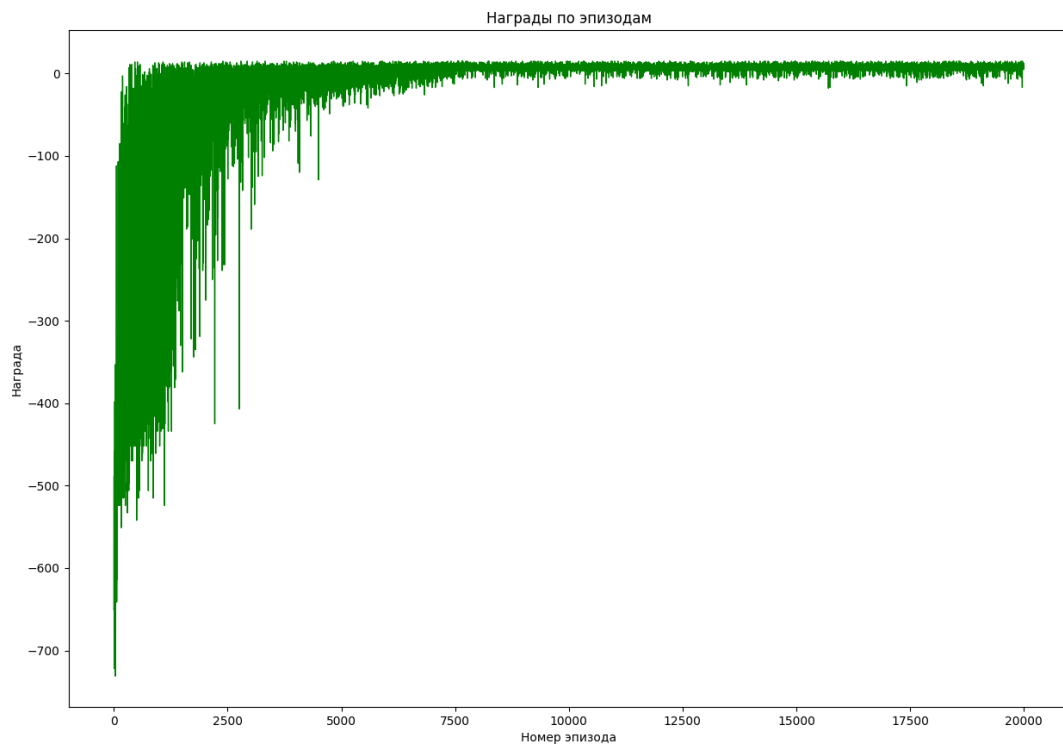


Рис. 1 – Награды по этапам SARSA.

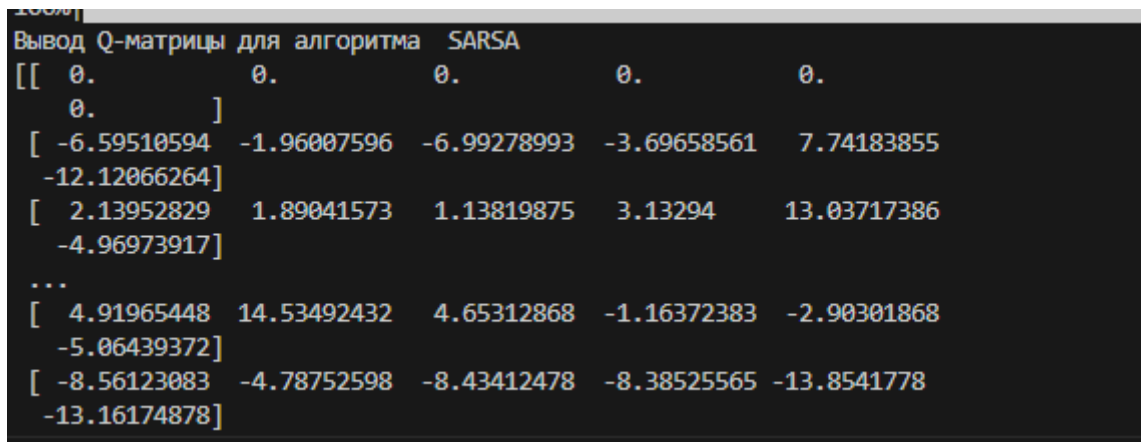


Рис. 2 – Q-матрица SARSA.

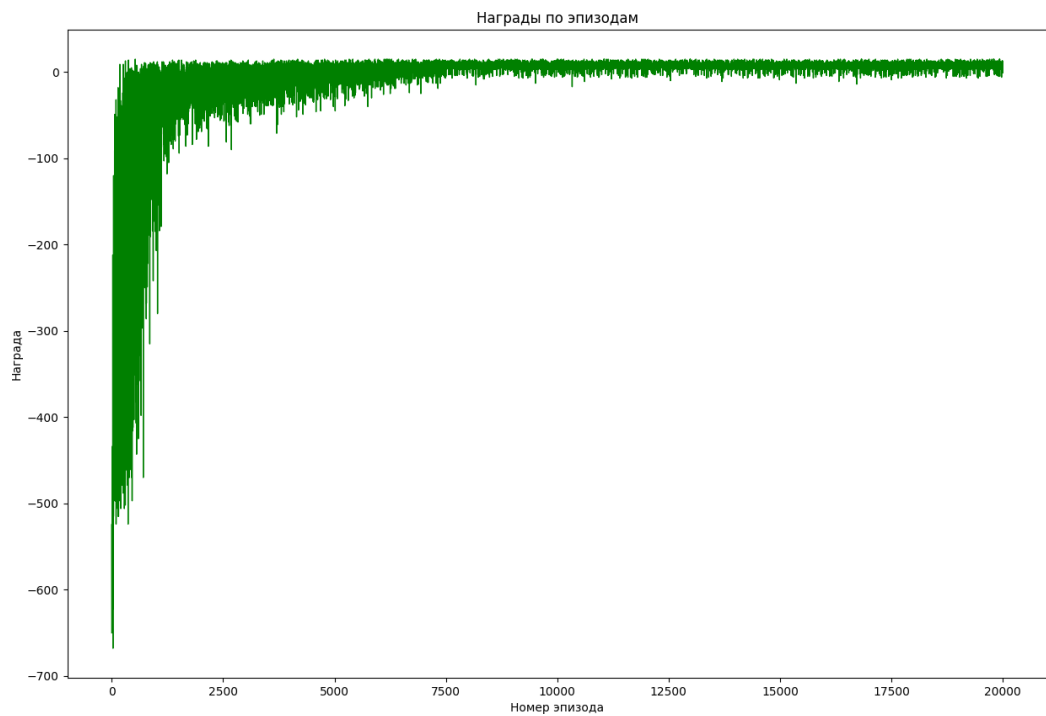


Рис. 3 – Награды по этапам Q-обучение.

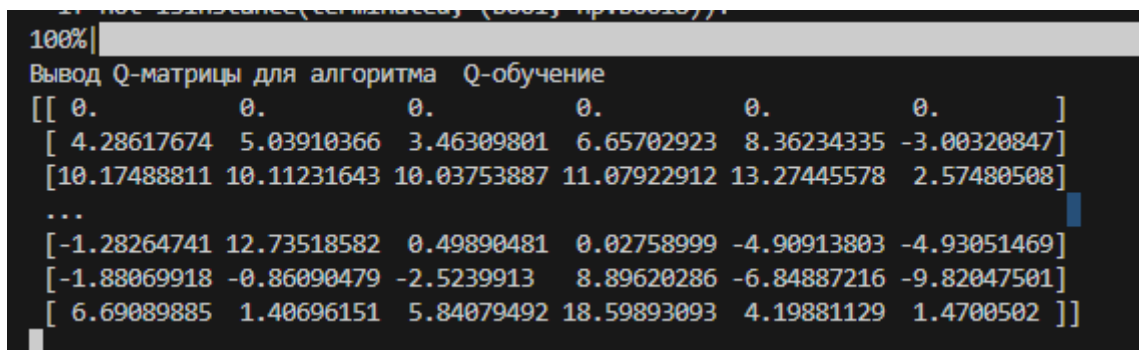


Рис. 4 – Q-матрица Q-обучения.

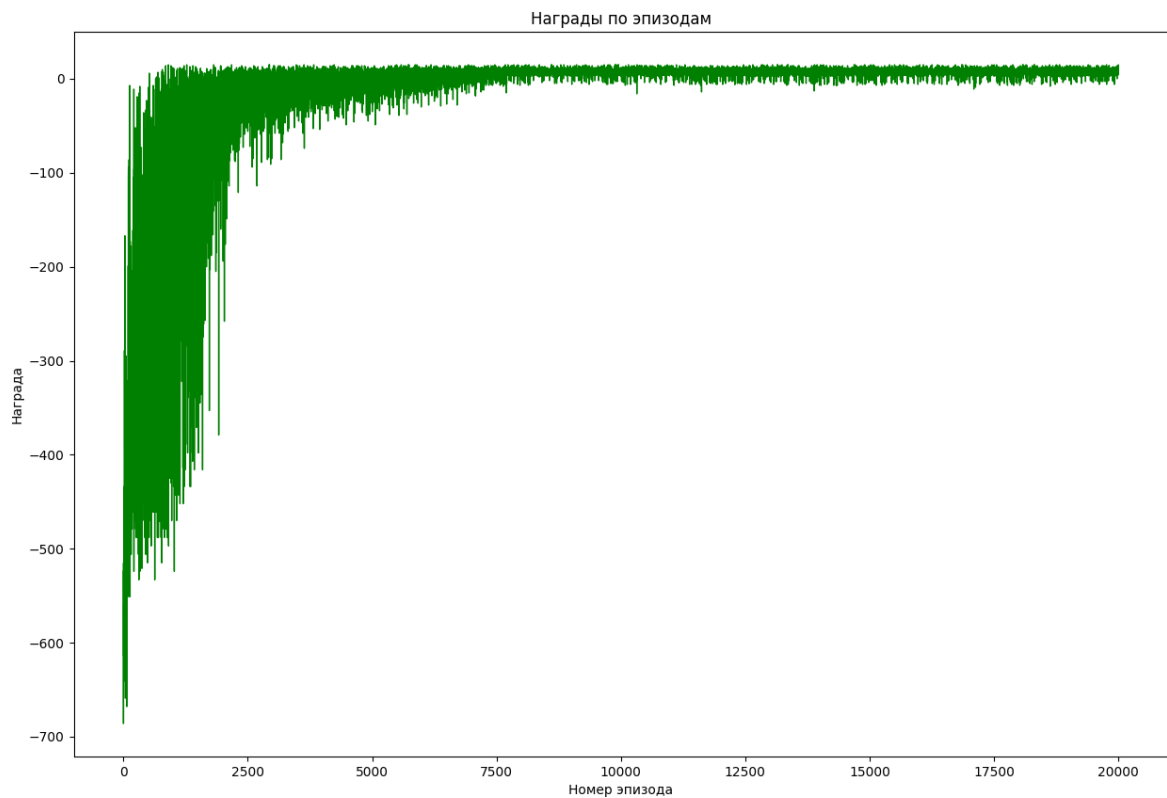


Рис. 5 – Награды по этапам dQ-обучение.

```

Вывод Q-матриц для алгоритма Двойное Q-обучение
Q1
[[ 0.         0.         0.         0.         0.
  0.         ]
 [ 0.4956301  2.01616985 -0.87676916 -1.03059973  8.36234335
 -4.40342241]
 [ 5.58468476  7.83243378  1.08344404  6.94931956 13.27445578
 -2.14348329]
 ...
 [ -1.41111304 10.63498294 -1.7376361  -1.21891579 -3.02222303
 -2.29858685]
 [ -4.08853292 -4.88230488 -4.55711092  2.70985571 -10.15945583
 -9.65504553]
 [ 1.40885897  1.98958961  3.8739806  18.47760311  0.65083876
 0.63652395]]
Q2
[[ 0.         0.         0.         0.         0.         0.         ]
 [-0.3270464  3.49955587 -3.33440988  1.05864569  8.36234335 -9.95144362]
 [ 1.33739738  6.88978543  2.8069611  7.18565328 13.27445578 -0.88784056]
 ...
 [-1.49141743  7.32475969 -1.22470837  0.83584724 -5.72093994 -3.7601113 ]
 [-4.57704201 -4.06435558 -4.26295557 -0.64485005 -9.27489379 -6.2435429 ]
 [ 1.33162216  4.41925175  5.21988787 18.33312556 -0.67761377  0.77036741]]

```

Рис. 6 – Q-матрицы dQ-обучение.

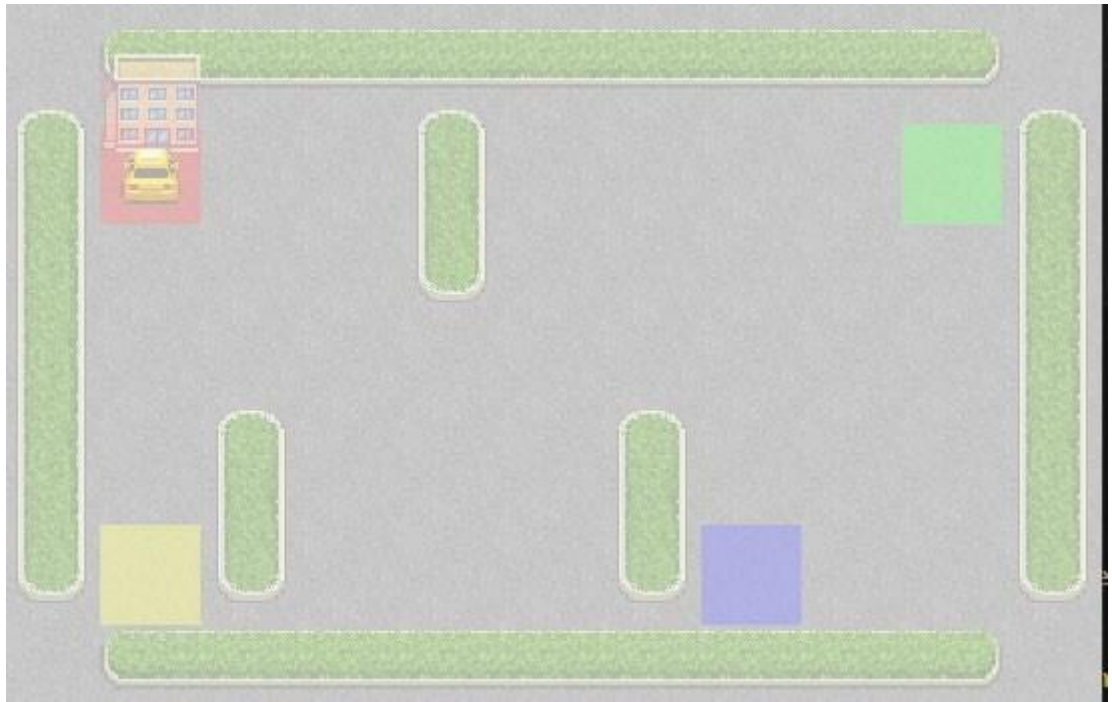


Рис. 7 – конечное состояние.

Вывод:

В ходе выполнения лабораторной работы мы ознакомились с базовыми методами обучения с подкреплением на основе временных различий.

Метод SARSA оказался самым быстрым, 2000 итераций в сек. Против 1900 и 1700 итераций в Q и dQ обучении. Вероятно, это связано с $\max()$ в Q и появлением промахов кэша в dQ.

Для настолько простой симуляции все алгоритмы имеют похожее время схождения, но Q и dQ имеют больший темп начального схождения.