|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования  Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **"Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)"**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА \_\_\_\_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)\_\_\_\_\_\_\_\_\_

ОТЧЕТ

# Лабораторная работа №4

«Алгоритм Policy Iteration»

по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬ:

группа ИУ5-23М

Гаврилов Л.Я.

ФИО

подпись

" " 2023 г.

ПРЕПОДАВАТЕЛЬ:

Москва - 2023

Гапанюк Ю.Е.

ФИО

подпись

" " 2023 г.

**Цель работы**

Ознакомление с базовыми методами обучения с подкреплением.

**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](https://www.gymlibrary.dev/) (или аналогичной библиотеки).

**Выполнение**

Для реализации алгоритма Policy iteration была выбрана среда Taxi из библиотеки Gym. Агент может находится в 25 позициях, пассажир может находится в 5 позициях, и 4 позиции для места назначения = 25\*5\*4= 500 состояний системы.

**Текст программы:**

**Policy\_iteration.py**

import gym

import numpy as np

import matplotlib.pyplot as plt

from pprint import pprint

class PolicyIterationAgent:

    '''

    Класс, эмулирующий работу агента

    '''

    def \_\_init\_\_(self, env):

        self.env = env

        # Пространство состояний

        self.observation\_dim = 500

        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)), 0.16667)

        # 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\_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 выполнился 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', render\_mode='human')

    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()

**Flake.py**

import gym

import numpy as np

import time

import matplotlib.pyplot as plt

from pprint import pprint

def main():

    state, action = 0, 0

    env = gym.make("Taxi-v3")

    print('Пространство состояний:')

    pprint(env.observation\_space)

    print()

    print('Пространство действий:')

    pprint(env.action\_space)

    print()

    print('Диапазон наград:')

    pprint(env.reward\_range)

    print()

    print('Вероятности для 0 состояния и 0 действия:')

    pprint(env.P[state][action])

    print()

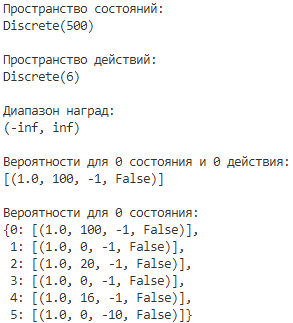
    print('Вероятности для 0 состояния:')

    pprint(env.P[state])

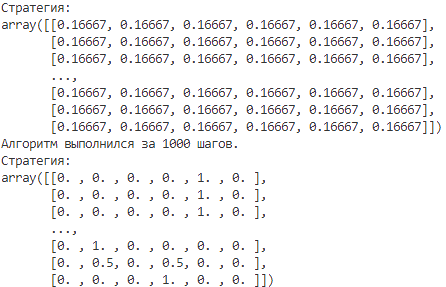
if \_\_name\_\_ == '\_\_main\_\_':

    main()

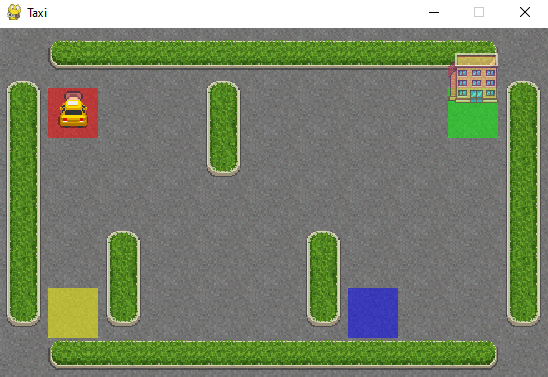
**Результат выполнения flake.py**



**Результат выполнения policy\_iteration.py. Начальная и конечная стратегии:**



Пример работы агента:



**Вывод**

В ходе выполнения лабораторной работы мы ознакомились с базовыми методами обучения с подкреплением, а именно алгоритмом policy iteration.