Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**«Методы машинного обучения в автоматизированных системах обработки информации и управления»**

**Лабораторная работа №4**

**«Реализация алгоритма Policy Iteration»**

**ИСПОЛНИТЕЛЬ:**

Демирев Н.К.

Группа ИУ5-21М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

"\_\_"\_\_\_\_\_\_\_\_\_\_\_2023 г.

Москва 2023

## Задание

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).

## Листинг

import gymnasium as gym

import numpy as np

import os

import pygame

from pprint import pprint

from enum import Enum

from tqdm import tqdm

import asyncio

NAME = 'Taxi-v3'

# Класс-перечисление действий

class Action(Enum):

    DOWN = 0

    UP = 1

    RIGHT = 2

    LEFT = 3

    PICKUP = 4

    DROP = 5

# Класс, эмулирующий работу агента

class PolicyIterationAgent:

    def \_\_init\_\_(self, env):

        self.env = env

        # Пространство состояний

        self.observation\_dim = 500

        # Массив действий в соответствии с документацией

        # https://gymnasium.farama.org/environments/toy\_text/taxi/

        self.actions\_variants = np.array([

            Action.DOWN,

            Action.UP,

            Action.RIGHT,

            Action.LEFT,

            Action.PICKUP,

            Action.DROP

            ])

        # Задание стратегии (политики)

        # Карта 5х5 и 6 возможных действия

        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(' Strategy matrix')

        pprint(self.policy\_probs)

    def policy\_evaluation(self): #Оценивание политики(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, bar : tqdm): #Основная реализация алгоритма

        policy\_stable = False

        bar.total = cnt

        for i in range(1, cnt + 1):

            self.state\_values = self.policy\_evaluation()

            self.policy\_probs = self.policy\_improvement()

            bar.update()

        print(f' Steps\n')

def play\_agent(agent):

    env2 = gym.make(NAME, 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(NAME)

    env.reset()

    # Обучение агента

    agent = PolicyIterationAgent(env)

    agent.print\_policy()

    bar = tqdm(position=1, leave=False, bar\_format=' {l\_bar}{bar:20}{r\_bar}{bar:-10b}', colour='CYAN')

    agent.policy\_iteration(1000, bar)

    bar.close()

    agent.print\_policy()

    # Проигрывание сцены для обученного агента

    play\_agent(agent)

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

    main()

## Экранные формы



