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:

Васильев Д.А.
Группа ИУ5-21М

_____

"__"_____2023 г.

Москва 2023

1. 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).

2. 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 5x5 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.sta
te_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()
```

3. Экранные формы

```
(venv) PS G:\repos\MMO\4 lab> python .\emulate.py
 Strategy matrix
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, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25, 0.25, 0.25]])

Steps|                    | 1000/1000 [00:22<00:00, 43.90it/s]


Strategy matrix
array([[0. , 0. , 0. , 0. , 1. , 0. ],
       [0. , 0. , 0. , 0. , 1. , 0. ],
       [0. , 0. , 0. , 0. , 1. , 0. ],
       ...,
       [0. , 1. , 0. , 0. , 0. , 0. ],
       [0. , 0.5, 0. , 0.5, 0. , 0. ],
       [0. , 0. , 0. , 1. , 0. , 0. ]])
```