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: Чупис С.А.

ФИО

группа \_ИУ5-21М

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 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.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]+proba bility\*(reward+self.gamma\*self.state\_values[next\_state])

[state,:]))

# Находим лучшие индексы

bestActionIndex=np.where(qvaluesMatrix[state,:]==np.max(qvaluesMatrix

# 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') 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:

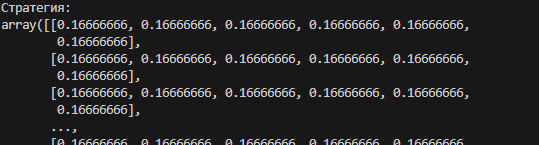


Рис. 1 – начальная стратегия.

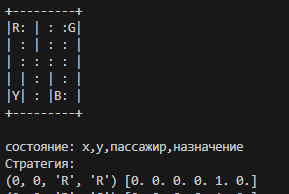


Рис. 2 – Модифицированный вывод.

Фрагмент вывода стратегии:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (0, | 0, | 'R', | 'R') | [0. | 0. | 0. 0. | | 1. | 0.] |  |  |
| (0, | 0, | 'R', | 'G') | [0. | 0. | 0. 0. | | 1. | 0.] |  |  |
| (0, | 0, | 'R', | 'Y') | [0. | 0. | 0. 0. | | 1. | 0.] |  |  |
| (0, | 0, | 'R', | 'B') | [0. | 0. | 0. 0. | | 1. | 0.] |  |  |
| (0, | 0, | 'G', | 'R') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'G', | 'G') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'G', | 'Y') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'G', | 'B') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'Y', | 'R') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| (0, | 0, | 'Y', | 'G') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| (0, | 0, | 'Y', | 'Y') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| (0, | 0, | 'Y', | 'B') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| (0, | 0, | 'B', | 'R') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'B', | 'G') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'B', | 'Y') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'B', | 'B') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'T', | 'R') | [0. 0. | | 0. 0. | | 0. | 1.] |  |  |
| (0, | 0, | 'T', | 'G') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 0, | 'T', | 'Y') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| (0, | 0, | 'T', | 'B') | [0.5 0. | |  | 0.5 | 0. | 0. | 0. | ] |
| (0, | 1, | 'R', | 'R') | [0. 0. | | 0. 1. | | 0. | 0.] |  |  |
| (0, | 1, | 'R', | 'G') | [0. 0. | | 0. 1. | | 0. | 0.] |  |  |
| (0, | 1, | 'R', | 'Y') | [0. 0. | | 0. 1. | | 0. | 0.] |  |  |
| (0, | 1, | 'R', | 'B') | [0. 0. | | 0. 1. | | 0. | 0.] |  |  |
| (0, | 1, | 'G', | 'R') | [1. 0. | | 0. 0. | | 0. | 0.] |  |  |
| **. . .** | | | | | | | | | | | |
| (4, | 4, | 'R', | 'R') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'R', | 'G') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'R', | 'Y') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'R', | 'B') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'G', | 'R') | [0. | 1. 0. 0. | | | 0. | 0.] |  |  |
| (4, | 4, | 'G', | 'G') | [0. | 1. 0. 0. | | | 0. | 0.] |  |  |
| (4, | 4, | 'G', | 'Y') | [0. | 1. 0. 0. | | | 0. | 0.] |  |  |
| (4, | 4, | 'G', | 'B') | [0. | 1. 0. 0. | | | 0. | 0.] |  |  |
| (4, | 4, | 'Y', | 'R') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'Y', | 'G') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'Y', | 'Y') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'Y', | 'B') | [0. | 0.5 | | 0. | 0.5 0. | | 0. | ] |
| (4, | 4, | 'B', | 'R') | [0. | 0. 0. 1. | | | 0. | 0.] |  |  |
| (4, | 4, | 'B', | 'G') | [0. | 0. 0. 1. | | | 0. | 0.] |  |  |
| (4, | 4, | 'B', | 'Y') | [0. | 0. 0. 1. | | | 0. | 0.] |  |  |
| (4, | 4, | 'B', | 'B') | [0. | 0. 0. 1. | | | 0. | 0.] |  |  |
| (4, 4, 'T', 'R') [0. 0.5 0. 0.5 0. 0. ]  (4, 4, 'T', 'G') [0. 1. 0. 0. 0. 0.] | | | | | | | | | | | |

(4, 4, 'T', 'Y') [0. 0.5 0. 0.5 0. 0. ]

(4, 4, 'T', 'B') [0. 0. 0. 1. 0. 0.]

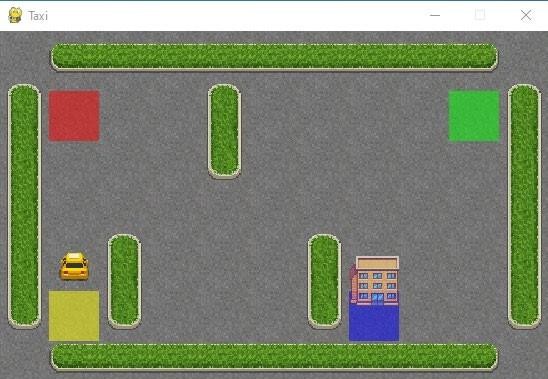


Рис. 3 – работа программы.

# 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 взятие и высадку пассажира вне ожидаемой зоны.

Таким образом, все переходы будут равнозначны до момента нахождения пассажира, и его случайной доставки.

Оптимизация стратегии начинается с +20 для состояний, в которых пассажир в такси, и он на нужной точке высадки (результат действия v = -10+20 = 10), после этого соседние состояния, имеющие возможность попасть в это, будут направлены в него. Технически решение достижимо за 12 итераций для 1 точки высадки.