Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика, искусственный интеллект и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет по лабораторной работе №4**

**по дисциплине «Методы машинного обучения»**

Реализация алгоритма Policy Iteration .

(тема работы)

ИСПОЛНИТЕЛЬ:

Пасатюк А.Д.

группа ИУ5-23М

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.А.

Москва, 2023

**Цель работы**

Ознакомление с базовыми методами обучения с подкреплением.

**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).

**Выполнение**

Реализуем алгоритм Policy Iteration для среды Toy Text / CliffWalking-v0.

Код программы:

**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 = 48  
 *# Массив действий в соответствии с документацией  
 # https://www.gymlibrary.dev/environments/toy\_text/frozen\_lake/* self.actions\_variants = np.array([0, 1, 2, 3])  
 *# Задание стратегии (политики)  
 # Карта 4х4 и 4 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(**'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(**'CliffWalking-v0'**, render\_mode=**'human'**)  
 state = env2.reset()  
 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(**'CliffWalking-v0'**)  
 env.reset()  
 *# Обучение агента* agent = PolicyIterationAgent(env)  
 agent.print\_policy()  
 agent.policy\_iteration(1500)  
 agent.print\_policy()  
 *# Проигрывание сцены для обученного агента* play\_agent(agent)  
 *#print(agent.policy\_probs)***if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()

Результат работы программы:

Стратегия:

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],

[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],

[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],

[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],

[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],

[0.25, 0.25, 0.25, 0.25],

[0.25, 0.25, 0.25, 0.25],

[0.25, 0.25, 0.25, 0.25]])

Алгоритм выполнился за 1500 шагов.

Стратегия:

array([[0.25 , 0.25 , 0.25 , 0.25 ],

[0.5 , 0. , 0. , 0.5 ],

[0. , 0. , 0. , 1. ],

[0. , 0.5 , 0.5 , 0. ],

[0. , 1. , 0. , 0. ],

[0.5 , 0.5 , 0. , 0. ],

[0.5 , 0. , 0. , 0.5 ],

[0. , 0. , 0. , 1. ],

[0. , 0. , 0.5 , 0.5 ],

[0. , 1. , 0. , 0. ],

[0.5 , 0.5 , 0. , 0. ],

[0.25 , 0.25 , 0.25 , 0.25 ],

[0.5 , 0. , 0. , 0.5 ],

[0.5 , 0. , 0. , 0.5 ],

[0. , 0.5 , 0.5 , 0. ],

[0. , 0.5 , 0.5 , 0. ],

[0. , 0. , 1. , 0. ],

[0.5 , 0. , 0.5 , 0. ],

[0.5 , 0. , 0.5 , 0. ],

[0. , 0. , 1. , 0. ],

[0. , 0. , 0.5 , 0.5 ],

[0. , 0. , 0.5 , 0.5 ],

[0.5 , 0.5 , 0. , 0. ],

[0.5 , 0.5 , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[0. , 1. , 0. , 0. ],

[0. , 1. , 0. , 0. ],

[0. , 1. , 0. , 0. ],

[0. , 1. , 0. , 0. ],

[0. , 0.5 , 0. , 0.5 ],

[0. , 0.5 , 0. , 0.5 ],

[0. , 0. , 0. , 1. ],

[0. , 0. , 0. , 1. ],

[0. , 0. , 0. , 1. ],

[0. , 0. , 0. , 1. ],

[1. , 0. , 0. , 0. ],

[0.33333333, 0. , 0.33333333, 0.33333333],

[0.5 , 0. , 0. , 0.5 ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[1. , 0. , 0. , 0. ],

[0.5 , 0.5 , 0. , 0. ],

[0.33333333, 0.33333333, 0.33333333, 0. ]])