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:	<u> Лу Сяои</u> ФИО
группа ИУ5И-22М	
	подпись
	"1" <u>Июнь</u> 2023 г.
ПРЕПОДАВАТЕЛЬ:	AUO
	ФИО
	подпись
	" " 2023 г.

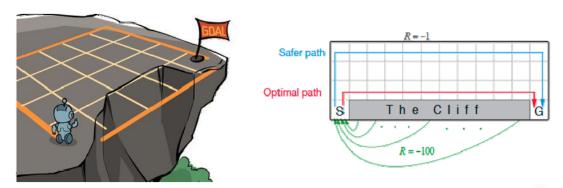
Москва - 2023

описание задания

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).

текст программы и экранные формы с примерами выполнения программы.

Я выбрала среду Cliff Walking.



Это мир с сеткой 4х12, где каждая сетка представляет собой состояние. Начальной точкой интеллекта является состояние в левом нижнем углу, а целью - состояние в правом нижнем углу. Интеллект может выполнять 4 действия в каждом состоянии: вверх, вниз, влево и вправо. Если интеллект совершает действие и касается пограничной стены, состояние не меняется, в противном случае он переходит в следующее состояние соответственно. В окружении есть участок обрыва, падение в который или достижение целевого состояния завершает действие и возвращает в начальную точку, т.е. падение в обрыв или достижение целевого состояния является конечным состоянием. Награда за каждый шаг равна -1, награда за падение с обрыва равна -100, а награда за достижение конечного состояния равна 0.

```
import copy
class CliffWalkingEnv:
   """ 悬崖漫步环境"""
   def __init__(self, ncol=12, nrow=4):
       self.ncol = ncol # 定义网格世界的列
       self.nrow = nrow # 定义网格世界的行
       # 转移矩阵 P[state][action] = [(p, next_state, reward, done)]包含下一个状态和奖励
       self.P = self.createP()
   def createP(self):
       # 初始化
       P = [[[] for j in range(4)] for i in range(self.nrow * self.ncol)]
       # 4 种动作, change[0]:上,change[1]:下, change[2]:左, change[3]:右。坐标系原点(0,0)
       # 定义在左上角
       change = [[0, -1], [0, 1], [-1, 0], [1, 0]]
       for i in range(self.nrow):
           for j in range(self.ncol):
              for a in range(4):
                  # 位置在悬崖或者目标状态,因为无法继续交互,任何动作奖励都为 0
                  if i == self.nrow - 1 and j > 0:
                     P[i * self.ncol + j][a] = [(1, i * self.ncol + j, 0,
                                               True)]
                     continue
```

```
# 其他位置
                  next_x = min(self.ncol - 1, max(0, j + change[a][0]))
                  next_y = min(self.nrow - 1, max(0, i + change[a][1]))
                  next_state = next_y * self.ncol + next_x
                  reward = -1
                  done = False
                  # 下一个位置在悬崖或者终点
                  if next_y == self.nrow - 1 and next_x > 0:
                      done = True
                      if next x != self.ncol - 1: # 下一个位置在悬崖
                         reward = -100
                  P[i * self.ncol + j][a] = [(1, next_state, reward, done)]
       return P
class PolicyIteration:
   """ 策略迭代算法 """
   def __init__(self, env, theta, gamma):
       self.env = env
       self.v = [0] * self.env.ncol * self.env.nrow # 初始化价值为 0
       self.pi = [[0.25, 0.25, 0.25, 0.25]
                 for i in range(self.env.ncol * self.env.nrow)] # 初始化为均匀随机策略
       self.theta = theta # 策略评估收敛阈值
       self.gamma = gamma # 折扣因子
   def policy_evaluation(self): # 策略评估
       cnt = 1 # 计数器
       while 1:
          max_diff = 0
          new_v = [0] * self.env.ncol * self.env.nrow
           for s in range(self.env.ncol * self.env.nrow):
              qsa list = [] # 开始计算状态 s 下的所有 Q(s,a)价值
              for a in range(4):
                  qsa = 0
                  for res in self.env.P[s][a]:
                      p, next_state, r, done = res
                      qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
                      # 本章环境比较特殊,奖励和下一个状态有关,所以需要和状态转移概率相乘
                  qsa_list.append(self.pi[s][a] * qsa)
              new v[s] = sum(qsa list) # 状态价值函数和动作价值函数之间的关系
              max_diff = max(max_diff, abs(new_v[s] - self.v[s]))
           self.v = new_v
           if max diff < self.theta: break # 满足收敛条件,退出评估迭代
          cnt += 1
       print("Strategy evaluation completed after %d round" % cnt)
   def policy_improvement(self): # 策略提升
       for s in range(self.env.nrow * self.env.ncol):
           qsa_list = []
           for a in range(4):
              qsa = 0
              for res in self.env.P[s][a]:
                  p, next_state, r, done = res
                  qsa += p * (r + self.gamma * self.v[next_state] * (1 - done))
              qsa_list.append(qsa)
           maxq = max(qsa_list)
          cntq = qsa_list.count(maxq) # 计算有几个动作得到了最大的 Q 值
           # 让这些动作均分概率
          self.pi[s] = [1 / cntq if q == maxq else 0 for q in qsa_list]
       print("Strategy enhancement completed")
       return self.pi
   def policy_iteration(self): # 策略迭代
       while 1:
           self.policy_evaluation()
           old_pi = copy.deepcopy(self.pi) # 将列表进行深拷贝,方便接下来进行比较
          new_pi = self.policy_improvement()
          if old_pi == new_pi: break
```

```
def print agent(agent, action meaning, disaster=[], end=[]):
   print("Status Value: ")
   for i in range(agent.env.nrow):
       for j in range(agent.env.ncol):
           # 为了输出美观,保持输出6个字符
           print('%6.6s' % ('%.3f' % agent.v[i * agent.env.ncol + j]), end=' ')
       print()
   print("Strategies: ")
   for i in range(agent.env.nrow):
       for j in range(agent.env.ncol):
           # 一些特殊的状态,例如悬崖漫步中的悬崖
           if (i * agent.env.ncol + j) in disaster:
               print('****', end=' ')
           elif (i * agent.env.ncol + j) in end: # 目标状态
               print('EEEE', end=' ')
               a = agent.pi[i * agent.env.ncol + j]
               pi_str = ''
               for k in range(len(action_meaning)):
                   pi_str += action_meaning[k] if a[k] > 0 else 'o'
               print(pi_str, end=' ')
       print()
```

```
env = CliffWalkingEnv()
action_meaning = ['^', 'v', '<', '>']
theta = 0.001
gamma = 0.9
agent = PolicyIteration(env, theta, gamma)
agent.policy_iteration()
print_agent(agent, action_meaning, list(range(37, 47)), [47])
```

Печатает значение текущей политики в каждом состоянии и действия, которые предприме т интеллект. Для выводимых действий мы используем ^o<o для обозначения равной вероя тности принятия как левых, так и восходящих действий, и ооо> для обозначения того, что в текущем состоянии принимаются только правые действия.

```
Strategy evaluation completed after 60 round
Strategy enhancement completed
                Strategy evaluation completed after 72 round
                Strategy enhancement completed
                Strategy evaluation completed after 44 round
                Strategy enhancement completed
                Strategy evaluation completed after 12 round
                Strategy enhancement completed
                Strategy evaluation completed after 1 round
                Strategy enhancement completed
                Status Value:
                -7.712 -7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710
                -7.458 -7.176 -6.862 -6.513 -6.126 -5.695 -5.217 -4.686 -4.095 -3.439 -2.710 -1.900
                 -7.\ 176\ \ -6.\ 862\ \ -6.\ 513\ \ -6.\ 126\ \ -5.\ 695\ \ -5.\ 217\ \ -4.\ 686\ \ -4.\ 095\ \ -3.\ 439\ \ -2.\ 710\ \ -1.\ 900\ \ -1.\ 000
                 -7.\ 458\quad 0.\ 000\quad 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 0.\ 000\ 
                Strategies:
                000 000 000 000 000 000 000 000 000 000 000 000
                000 000 000 000 000 000 000 000 000 000 000 000
```