

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:

Лу Сяои
ФИО

группа ИУ5И-22М

подпись

"1" Июнь 2023 г.

ПРЕПОДАВАТЕЛЬ:

ФИО

подпись

" " _____ 2023 г.

Москва - 2023

```

        # 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=' ')
            else:
                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 предпримет интеллект. Для выводимых действий мы используем $\wedge o < o$ для обозначения равной вероятности принятия как левых, так и восходящих действий, и $ooo >$ 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 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
Strategies:
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo>
ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo> ovo>
ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo> ooo>
^ooo **** **** **** **** **** **** **** **** **** **** EEEE
```