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

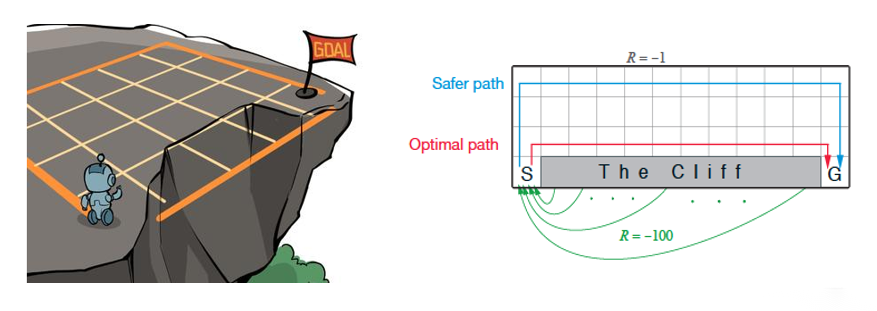
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **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 4x12, 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=' ')

            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 ^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.

