# MachineLearningHomework1_ReinforcementLearning

201250068 陈骏

## 环境配置

```
Python 3.7
Package:
    copy
    gym
```

## 项目代码

### 代码结构

```
RL.py
|
|
|---class CliffWalkingEnv # 悬崖漫步的环境
|   |
|   |--- __init__(self, ncol=12, nrow=4) # 初始化悬崖大小及转移矩阵内容
|   |
|   |--- createP(self) # 创建转移矩阵P
|
|
|---class PolicyIteration # 决策迭代
|   |
|   |--- __init__(self, env, theta, gamma) # 初始化决策对应的悬崖环境及决策学习参数
|   |
|   |--- policy_improvement(self) # 优化一次决策，计算每个节点向周围可能节点转移的概率
|   |
|   |--- policy_evaluation(self) # 对当前的转移矩阵进行计算每个点的v值，进行价值迭代
|   |
|   |--- policy_iteration(self): # 迭代转移策略
|
|
|--- print_agent(agent, action_meaning, disaster=[], end=[]) # 打印迭代的最终结果
```

### RL实现

策略迭代，每次更新价值之后，对于每个节点，计算每种可能的转移策略下，下一步的价值，在所有可能的转移结果下，对于一个或多个最大价值，其中，每次非终止转移Reward为-1，终止转移Reward为-100，其对应的转移策略平分该节点的转移概率。代码中由policy_improvement实现。

价值迭代，每次策略迭代后，对于每个节点，其价值等于其可能的转移策略（转移概率大于1）的下一步价值与反馈Reward的和的加权，权重为转移概率。代码中由policy_evaluation实现。

#### RL.py代码内容

```python
import copy


class CliffWalkingEnv:
    def __init__(self, ncol=12, nrow=4):
        self.ncol = ncol
        self.nrow = nrow

        self.P = self.createP()

    def createP(self):
        ##
        P = [[[] for i in range(4)] for i in range(self.nrow * self.ncol)]
        # 对应action_meaning
        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):
                    if i == self.nrow - 1 and j > 0:
                        # 一个大小为四的数组，四个意义分别代表
                        # 转移概率， 下一步的状态点(当前condition下为不动，即本省)， 反馈值， 是否结束
                        P[i * self.ncol + j][a] = [(1, i * self.ncol + j, 0, True)]
                        continue
                    # 对于某一个策略，下一步的移动目标为next_state
                    # change 为 （x, y）
                    # min 与 max 用于限定下一个状态不出当前边界
                    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
                    # 下一个位置是悬崖或者重点
                    if next_y == (self.nrow - 1) and next_x > 0:
                        # 设定为最下面一排中间一排是悬崖，现在已经到了最下面一排并且不在最一开始的地方
                        done = True
                        if next_x != self.ncol - 1:
                            reward = -100
                    P[i * self.ncol + j][a] = [(1, next_state, reward, done)]
        return P


class PolicyIteration:

    def __init__(self, env, theta, gamma):
        self.env = env
        self.v = [0] * self.env.ncol * self.env.nrow
        # 状态价值矩阵，每一个格子的四种策略都有值
        self.pi = [[0.25, 0.25, 0.25, 0.25] for i in range(self.env.ncol * self.env.nrow)]
        self.theta = theta
        self.gamma = gamma

    def policy_improvement(self):
        for s in range(self.env.nrow * self.env.ncol):
```

```
            qsa_list = []
            # 对于每个点，计算当前节点的四个策略合并的下一步的状态权重作为当前节点的权值
            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))
                qsa_list.append(qsa)
            maxq = max(qsa_list)
            cntq = qsa_list.count(maxq)
           # 所有最大权值的可能转移策略平分转移概率
            self.pi[s] = [1 / cntq if q == maxq else 0 for q in qsa_list]
        print("策略提升完成")
        return self.pi

    def policy_evaluation(self):
        cnt = 1
        while True:
            max_diff = 0
            # 新的权值矩阵
            new_v = [0] * self.env.ncol * self.env.nrow
            for s in range(self.env.ncol * self.env.nrow):
                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(self.pi[s][a] * qsa)
                new_v[s] = sum(qsa_list) # 对四个可能的下一步节点进行求和
                # 计算最大的差值，当最大的权值差值也小于，即所有节点的权值变动都小于theme
                max_diff = max(max_diff, abs(new_v[s] - self.v[s]))
            self.v = new_v
            if max_diff < self.theta:
                break
            cnt += 1
        print("策略评估迭代" + str(cnt) + "轮后完成")

    def policy_iteration(self):
        # 直到迭代一次后，所有节点的转移方向即概率全部相同停止
        while True:
            self.policy_evaluation()
            old_pi = copy.deepcopy(self.pi)
            new_pi = self.policy_improvement()
            if old_pi == new_pi:
                break


def print_agent(agent, action_meaning, disaster=[], end=[]):
    print("状态价值：")
    for i in range(agent.env.nrow):
        for j in range(agent.env.ncol):
            print("%6.6s" % ("%.3f" % agent.v[i * agent.env.ncol + j]), end=' ')
        print()

    print("策略：")
```

```
    for i in range(agent.env.nrow):
        for j in range(agent.env.ncol):
            if (i * agent.env.ncol + j) in disaster:
                print("****", end=' ')
            elif (i * agent.env.ncol + j) in end:
                print(" End", 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 '_'
                print(pi_str, end=' ')
        print()

env = CliffWalkingEnv()
action_meaning = ['↑', '↓', '←', '→']
theta = 0.001
gamma = 0.9
agent = PolicyIteration(env, theta, gamma)
agent.policy_iteration()
print(list(range(37, 47)))
print_agent(agent, action_meaning, list(range(37, 47)), [47])
print(agent.pi)
```

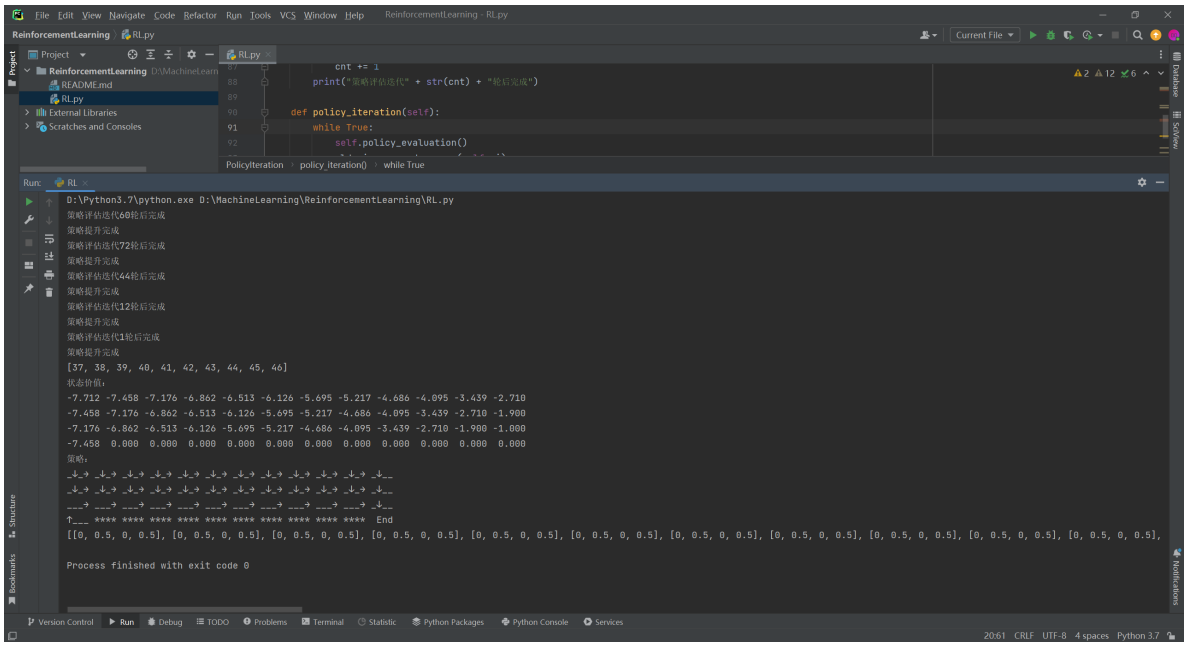
### RL_gym.py

```
import gym
from RL import print_agent, PolicyIteration
env = gym.make("FrozenLake-v1")
env = env.unwrapped
env.render()

holes = set()
ends = set()
for s in env.P:
    for a in env.P[s]:
        for s_ in env.P[s][a]:
            if s_[2] == 1.0:
                ends.add(s_[1])
            if s_[3] == True:
                holes.add(s_[1])
holes = holes -ends
print(holes)
print(ends)
for a in env.P[14]:
    print(env.P[14][a])
action_meaning = ['←', '↓', '→', '↑']
theta =1e-5
gamma = 0.9
agent = PolicyIteration(env, theta, gamma)
agent.policy_iteration()
print_agent(agent, action_meaning, holes, ends)
```

## 运行结果

# RL.py




# RL_gym.py