Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Лабораторная работа №4
по дисциплине «Методы машинного обучения»
«Алгоритм Policy Iteration»

ИСПОЛНИТЕЛЬ:

Алехин С.С.
Группа ИУ5-23М

____

ПРОВЕРИЛ:

Балашов А.М.

____

Москва, 2023

Задание

1. 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).

Lab4

June 21, 2023

[ ]:

```
! pip install gymnasium
! pip install pygame

import os
import pygame
import gymnasium as gym
import numpy as np
from pprint import pprint
```

```
Requirement already satisfied: gymnasium in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (0.28.1)
Requirement already satisfied: numpy>=1.21.0 in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (from gymnasium)
(1.25.0)
Requirement already satisfied: jax-jumpy>=1.0.0 in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (from gymnasium)
(1.0.0)
Requirement already satisfied: cloudpickle>=1.2.0 in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (from gymnasium)
(2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (from gymnasium)
(4.6.3)
Requirement already satisfied: farama-notifications>=0.0.1 in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (from gymnasium)
(0.0.4)
Requirement already satisfied: pygame in
/Users/seralekhin/BMSTU_Labs/.env/lib/python3.11/site-packages (2.4.0)
```

[ ]:

```
class PolicyIterationAgent:
    '''
       ,
    '''
    def __init__(self, env):
        self.env = env
        #
        self.observation_dim = 4 * 12
        #
        # https://gymnasium.farama.org/environments/toy_text/taxi/
        self.actions_variants = np.array([0,1,2,3])
        # 0: Move south (down)
        # 1: Move north (up)
        # 2: Move east (right)
        # 3: Move west (left)
        # 4: Pickup passenger
        # 5: Drop off passenger
        #            (    )
        #     4 12   4
        self.policy_probs = np.full((self.observation_dim, len(self.
↪actions_variants)), 0.25)
        #               v(s)
        self.state_values = np.zeros(shape=(self.observation_dim))
        #
        self.maxNumberOfIterations = 1000
        self.theta=1e-6
        self.gamma=0.99

    def print_policy(self):
        '''

        '''
        print('     :')
        pprint(self.policy_probs)

    def policy_evaluation(self):
        '''

        '''
        #
        valueFunctionVector = self.state_values
        for iterations in range(self.maxNumberOfIterations):
            #
            valueFunctionVectorNextIteration=np.zeros(shape=(self.
↪observation_dim))
            #
            for state in range(self.observation_dim):
                #
                action_probabilities = self.policy_probs[state]
                #
                outerSum=0
                for action, prob in enumerate(action_probabilities):
                    innerSum=0
                    #
                    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):
                #
                valueFunctionVector=valueFunctionVectorNextIteration
                break
            valueFunctionVector=valueFunctionVectorNextIteration
        return valueFunctionVector


    def policy_improvement(self):
        '''

        '''
        qvaluesMatrix=np.zeros((self.observation_dim, len(self.
↪actions_variants)))
        improvedPolicy=np.zeros((self.observation_dim, len(self.
↪actions_variants)))
        #
        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])

            #
            bestActionIndex=np.where(qvaluesMatrix[state,:]==np.
↪max(qvaluesMatrix[state,:]))
            #
            improvedPolicy[state,bestActionIndex]=1/np.size(bestActionIndex)
        return improvedPolicy


    def policy_iteration(self, cnt):
        '''

        '''
        policy_stable = False
        for i in range(1, cnt+1):
            self.state_values = self.policy_evaluation()
            self.policy_probs = self.policy_improvement()
        print(f'              {i}    .')
```

```
[ ]: def play_agent(agent):
         env2 = gym.make('CliffWalking-v0', render_mode='human')
         state = env2.reset()[0]
         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
```

```
[ ]: #
     env = gym.make('CliffWalking-v0')
     env.reset()
     #
     agent = PolicyIterationAgent(env)
     agent.policy_iteration(1000)
     agent.print_policy()
     #
     play_agent(agent)
```

```
             1000    .
     :
array([[0.        , 0.5       , 0.5       , 0.        ],
       [0.33333333, 0.33333333, 0.33333333, 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.        , 1.        , 0.        ],
       [0.33333333, 0.        , 0.33333333, 0.33333333],
       [0.        , 0.        , 0.5       , 0.5       ],
       [0.        , 0.        , 1.        , 0.        ],
       [0.        , 0.5       , 0.5       , 0.        ],
       [0.        , 0.5       , 0.5       , 0.        ],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
[0.        , 0.33333333, 0.33333333, 0.33333333],
[0.        , 0.33333333, 0.33333333, 0.33333333],
[0.        , 0.33333333, 0.33333333, 0.33333333],
[0.        , 0.        , 0.5       , 0.5       ],
[0.        , 0.        , 0.5       , 0.5       ],
[0.        , 0.        , 1.        , 0.        ],
[0.        , 0.33333333, 0.33333333, 0.33333333],
[0.        , 0.5       , 0.        , 0.5       ],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.33333333, 0.33333333, 0.        , 0.33333333],
[0.        , 0.5       , 0.        , 0.5       ],
[0.        , 0.33333333, 0.33333333, 0.33333333],
[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.        ]])
```

Lab4

June 22, 2023

[ ]:
```python
import gymnasium as gym
import numpy as np
from pprint import pprint
```

[ ]:
```python
class PolicyIterationAgent:
    '''
       ,
    '''
    def __init__(self, env):
        self.env = env
        #
        self.observation_dim = 4 * 12
        #
        # https://gymnasium.farama.org/environments/toy_text/taxi/
        self.actions_variants = np.array([0,1,2,3])
        # 0: Move south (down)
        # 1: Move north (up)
        # 2: Move east (right)
        # 3: Move west (left)
        # 4: Pickup passenger
        # 5: Drop off passenger
        #            (     )
        #     4 12  4
        self.policy_probs = np.full((self.observation_dim, len(self.
 ↪actions_variants)), 0.25)
        #               v(s)
        self.state_values = np.zeros(shape=(self.observation_dim))
        #
        self.maxNumberOfIterations = 1000
        self.theta=1e-6
        self.gamma=0.99

    def print_policy(self):
        '''

        '''
        print('     :')
        pprint(self.policy_probs)

    def policy_evaluation(self):
        '''

        '''
        #
        valueFunctionVector = self.state_values
        for iterations in range(self.maxNumberOfIterations):
            #
            valueFunctionVectorNextIteration=np.zeros(shape=(self.observation_dim))
            #
            for state in range(self.observation_dim):
                #
                action_probabilities = self.policy_probs[state]
                #
                outerSum=0
                for action, prob in enumerate(action_probabilities):
                    innerSum=0
                    #
                    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):
                #
                valueFunctionVector=valueFunctionVectorNextIteration
                break
            valueFunctionVector=valueFunctionVectorNextIteration
        return valueFunctionVector


    def policy_improvement(self):
        '''

        '''
        qvaluesMatrix=np.zeros((self.observation_dim, len(self.actions_variants)))
        improvedPolicy=np.zeros((self.observation_dim, len(self.actions_variants)))
        #
        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])

            #
            bestActionIndex=np.where(qvaluesMatrix[state,:]==np.
↪max(qvaluesMatrix[state,:]))
            #
            improvedPolicy[state,bestActionIndex]=1/np.size(bestActionIndex)
        return improvedPolicy


    def policy_iteration(self, cnt):
        '''

        '''
        policy_stable = False
        for i in range(1, cnt+1):
            self.state_values = self.policy_evaluation()
            self.policy_probs = self.policy_improvement()
        print(f'              {i}    .')
```

[ ]:

```
def play_agent(agent):
    env2 = gym.make('CliffWalking-v0', render_mode='human')
    state = env2.reset()[0]
    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
```

[ ]:

```
#
env = gym.make('CliffWalking-v0')
env.reset()
#
agent = PolicyIterationAgent(env)
agent.policy_iteration(1000)
agent.print_policy()
#
play_agent(agent)
```

```
              1000    .
     :
array([[0.        , 0.5       , 0.5       , 0.        ],
       [0.33333333, 0.33333333, 0.33333333, 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.        , 1.        , 0.        ],
       [0.33333333, 0.        , 0.33333333, 0.33333333],
       [0.        , 0.        , 0.5       , 0.5       ],
       [0.        , 0.        , 1.        , 0.        ],
       [0.        , 0.5       , 0.5       , 0.        ],
       [0.        , 0.5       , 0.5       , 0.        ],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.        , 0.5       , 0.5       ],
       [0.        , 0.        , 0.5       , 0.5       ],
       [0.        , 0.        , 1.        , 0.        ],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [0.        , 0.5       , 0.        , 0.5       ],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.33333333, 0.33333333, 0.        , 0.33333333],
       [0.        , 0.5       , 0.        , 0.5       ],
       [0.        , 0.33333333, 0.33333333, 0.33333333],
       [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.        ]])
```

```
The Kernel crashed while executing code in the the current cell or a previous␣
  ↪cell. Please review the code in the cell(s) to identify a possible cause of␣
  ↪the failure. Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a>␣
  ↪for more info. View Jupyter <a href='command:jupyter.viewOutput'>log</a> for␣
  ↪further details.
```