ELEN_6885_HW4_Part_1_2_3

November 25, 2019

1 ELEN 6885 Reinforcement Learning Coding Assignment (Part 1, 2, 3)

1.1 Taxi Problem Overview

Please put your code into the block marked by: ############################## YOUR CODE STARTS HERE YOUR CODE ENDS HERE ############################## You should not edit anything outside of the block.

2 Playing with the environment

Run the cell below to get a feel for the environment by moving your agent(the taxi) by taking one of the actions at each step.

[1]:
```
from gym.wrappers import Monitor
import gym
import random
import numpy as np
```

[2]:
```
"""
You can test your game now.
Input range from 0 to 5:
    0 : South (Down)
    1 : North (Up)
    2 : East (Right)
    3 : West (Left)
    4: Pick up
    5: Drop off
    6: exit_game
"""
GAME = "Taxi-v3"
env = gym.make(GAME)
env = Monitor(env, "taxi_simple", force=True)
s = env.reset()
steps = 100
for step in range(steps):
    env.render()
```

```python
    action = int(input("Please type in the next action:"))
    if action==6:
        break
    s, r, done, info = env.step(action)
    print('state:',s)
    print('reward:',r)
    print('Is state terminal?:',done)
    print('info:',info)

# close environment and monitor
env.close()
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

Please type in the next action:0
state: 153
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
Please type in the next action:0
state: 253
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
Please type in the next action:2
```

```
state: 273
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
Please type in the next action:0
state: 373
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
Please type in the next action:0
state: 473
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
Please type in the next action:4
state: 477
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
```

```
|Y| : |B: |
+---------+
  (Pickup)
Please type in the next action:1
state: 377
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : |_: |
|Y| : |B: |
+---------+
  (North)
Please type in the next action:1
state: 277
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (North)
Please type in the next action:1
state: 177
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | :_: |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
Please type in the next action:1
state: 77
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
```

```
|R: | :_:G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
Please type in the next action:2
state: 97
reward: -1
Is state terminal?: False
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
Please type in the next action:5
state: 85
reward: 20
Is state terminal?: True
info: {'prob': 1.0}
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Please type in the next action:6
```

2.1 1.1 Incremental implementation of average

We've finished the incremental implementation of average for you. Please call the function to estimate with 1/step step size and fixed step size to compare the difference between these two on a simulated Bandit problem.

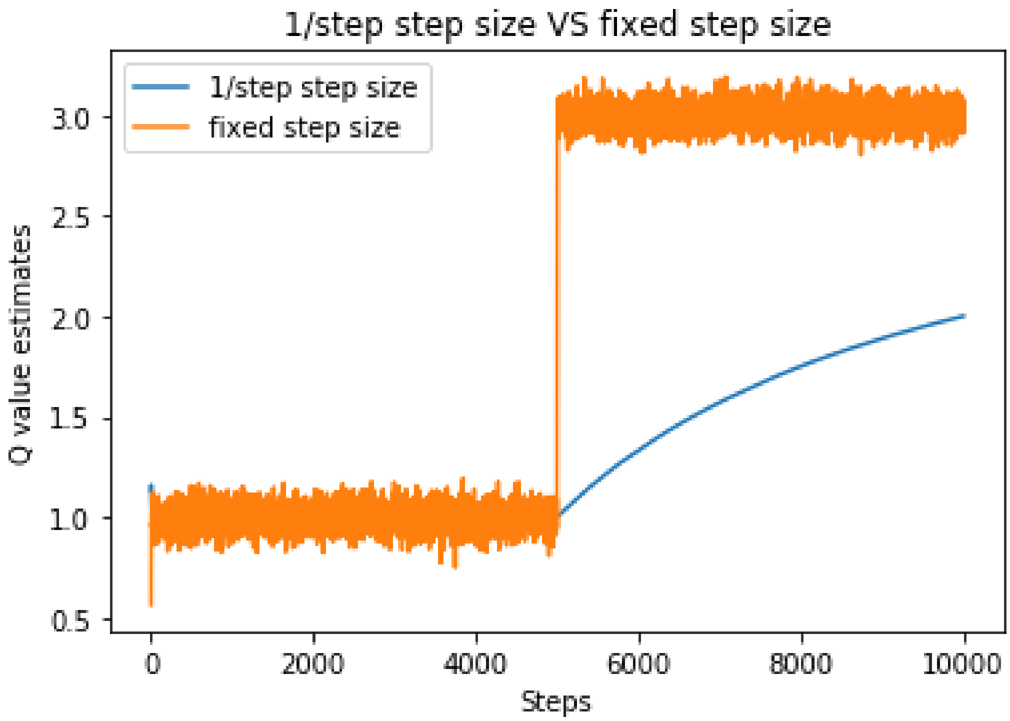
[3]:
```
def estimate(OldEstimate, StepSize, Target):
    '''An incremental implementation of average.
    OldEstimate : float
    StepSize : float
    Target : float
    '''
    NewEstimate = OldEstimate + StepSize * (Target - OldEstimate)
```

```
    return NewEstimate
```

[4]:
```
random.seed(6885)
numTimeStep = 10000
q_h = np.zeros(numTimeStep + 1) # Q Value estimate with 1/step step size
q_f = np.zeros(numTimeStep + 1) # Q value estimate with fixed step size
FixedStepSize = 0.5 #A large number to exaggerate the difference
for step in range(1, numTimeStep + 1):
    if step < numTimeStep / 2:
        r = random.gauss(mu = 1, sigma = 0.1)
    else:
        r = random.gauss(mu = 3, sigma = 0.1)

    #TIPS: Call function estimate defined in ./RLalgs/utils.py
    ############################
    # YOUR CODE STARTS HERE

    q_h[step] = estimate(q_h[step-1], 1/step, r)
    q_f[step] = estimate(q_f[step-1], FixedStepSize, r)

    # YOUR CODE ENDS HERE
    ############################

q_h = q_h[1:]
q_f = q_f[1:]
```

Plot the two Q value estimates. (Please include a title, labels on both axes, and legends)

[6]:
```
import matplotlib.pyplot as plt
############################
# YOUR CODE STARTS HERE

line1, = plt.plot([i for i in range(numTimeStep)],q_h, label="1/step step size")
line2, = plt.plot([i for i in range(numTimeStep)],q_f, label="fixed step size")

plt.legend()

plt.xlabel('Steps')
plt.ylabel('Q value estimates')
plt.title('1/step step size VS fixed step size')
plt.show()

# YOUR CODE ENDS HERE
############################
```



2.2 1.2 ϵ-Greedy for Exploration

In Reinforcement Learning, we are always faced with the dilemma of exploration and exploitation. ϵ-Greedy is a trade-off between them. You are supposed to implement Greedy and ϵ-Greedy. We combine these two policies in one function by treating Greedy as ϵ-Greedy where $\epsilon = 0$. Edit the function epsilon_greedy the following block.

[7]:
```
def epsilon_greedy(value, e, seed = None):
    '''
    Implement Epsilon-Greedy policy.

    Inputs:
    value: numpy ndarray
    A vector of values of actions to choose from
    e: float
    Epsilon
    seed: None or int
    Assign an integer value to remove the randomness

    Outputs:
    action: int
    Index of the chosen action
    '''
    assert len(value.shape) == 1
```

```
    assert 0 <= e <= 1

    if seed != None:
          np.random.seed(seed)

    ############################
    # YOUR CODE STARTS HERE

    if random.random() > e: # prob = 1-e
        return np.argmax(value)
    else: # prob = e
        return random.randrange(value.size)

    # YOUR CODE ENDS HERE
    ############################
    return action
```

[8]:
```
np.random.seed(6885) #Set the seed forreproducability
q = np.random.normal(0, 1, size = 5)
############################
# YOUR CODE STARTS HERE

greedy_action = epsilon_greedy(q, 0)
e_greedy_action = epsilon_greedy(q, 0.1)

# YOUR CODE ENDS HERE
############################
print('Values:')
print(q)
print('Greedy Choice =', greedy_action)
print('Epsilon-Greedy Choice =', e_greedy_action)
```

```
Values:
[ 0.61264537  0.27923079 -0.84600857  0.05469574 -1.09990968]
Greedy Choice = 0
Epsilon-Greedy Choice = 0
```

You should get the following results: Values: [0.61264537 0.27923079 -0.84600857 0.05469574 -1.09990968] Greedy Choice = 0 Epsilon-Greedy Choice = 0

2.3 1.3 Exploration VS. Exploitation

Try to reproduce Figure 2.2 (the upper one is enough) of the Sutton's book based on the experiment described in Chapter 2.3.

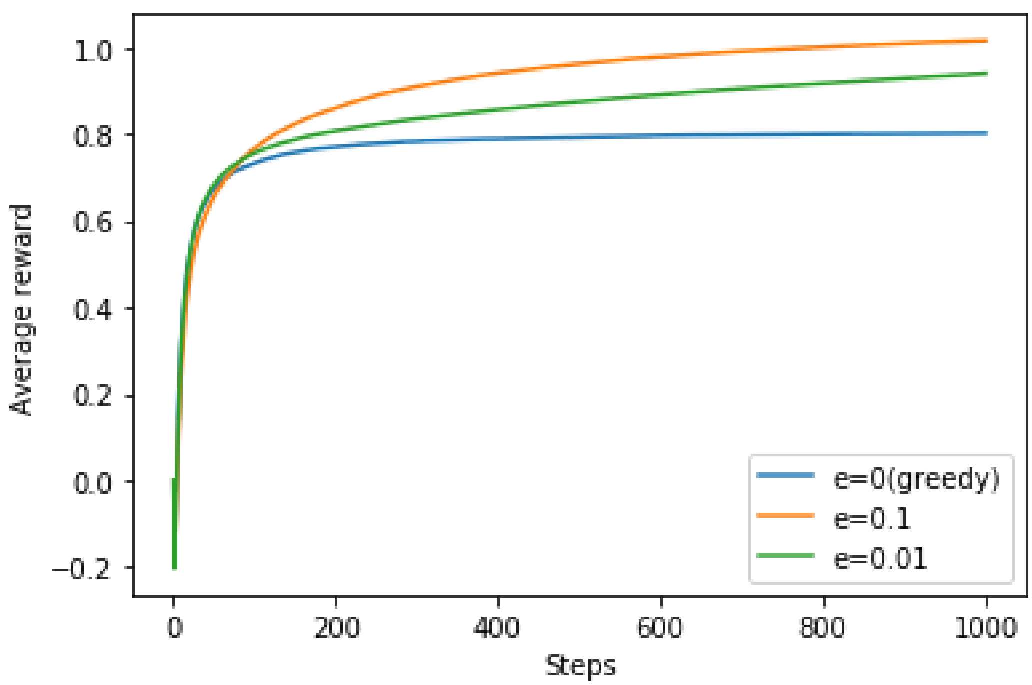
[9]:
```
# Do the experiment and record average reward acquired in each time step
############################
# YOUR CODE STARTS HERE
```

```python
def HW4_P1_3(T, k, runs, e, Q_true):

    # Because timestep < runs, to save memory, we store the average reward of each time step. Needs len(list)=1000
    result_reward = [0 for i in range(T)]                          # average reward of runs

    for run in range(runs):
        # one run
        average_reward = [0 for i in range(T)]     # average reward within each run
        choose_num = [0 for i in range(k)]         # the number each arm is chosen
        Q_estimate = np.random.normal(0, 1, size = k)                 # estimated action value based on sample averages

        for step in range(1, T):
            choose = epsilon_greedy(Q_estimate, e)# choose which arm
            choose_num[choose] += 1                # record the choose
            reward = random.gauss(mu = Q_true[choose], sigma = 1) # get reward
            # update the value estimate of the chosen arm
            Q_estimate[choose] = Q_estimate[choose] + (1/choose_num[choose]) * (reward - Q_estimate[choose])
            # update the average reward
            average_reward[step] = average_reward[step-1] + (1/step) * (reward - average_reward[step-1])

        # sum corresponding average_reward between runs
        result_reward = np.sum([average_reward,result_reward], axis = 0)

    for i in range(T):
        result_reward[i] = result_reward[i]/runs

    return result_reward


T = 1000                       # One run contain 1000 time steps
k = 10                         # 10-arm
runs = 2000                    # the Sutton conduct 2000 runs
Q_true = np.random.normal(0, 1, size = k) # true action value, which is constant for three different e

# greedy
e = 0                          # epsilon
result_0 = HW4_P1_3(T, k, runs, e, Q_true)
```

```
# epsilon equals 0.1
e = 0.1
result_01 = HW4_P1_3(T, k, runs, e, Q_true)
# epsilon equals 0.01
e = 0.01
result_001 = HW4_P1_3(T, k, runs, e, Q_true)

# YOUR CODE ENDS HERE
############################
```

[10]:

```
# Plot the average reward
############################
# YOUR CODE STARTS HERE

line0, = plt.plot([i for i in range(1,T+1)],result_0, label="e=0(greedy)")
line01, = plt.plot([i for i in range(1,T+1)],result_01, label="e=0.1")
line001, = plt.plot([i for i in range(1,T+1)],result_001, label="e=0.01")

plt.legend()

plt.xlabel('Steps')
plt.ylabel('Average reward')
plt.show()


# YOUR CODE ENDS HERE
############################
```



3 Question 2

In this question, you will implement the value iteration and policy iteration algorithms to solve the Taxi game problem

3.1 2.1 Model-based RL: value iteration

For this part, you need to implement the helper functions action_evaluation(env, gamma, v), and extract_policy(env, v, gamma) in utils.py. Understand action_selection(q) which we have implemented. Use these helper functions to implement the value_iteration algorithm below.

[11]:
```
import numpy as np
from helpers import utils
def value_iteration(env, gamma, max_iteration, theta):
    """
    Implement value iteration algorithm. You should use extract_policy to for␣
 ↪extracting the policy.

    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    gamma: float
            Discount factor.
    max_iteration: int
            The maximum number of iterations to run before stopping.
    theta: float
            Determines when value function has converged.
    Returns:
    ----------
    value function: np.ndarray
    policy: np.ndarray
    """
    V = np.zeros(env.nS)
    ############################
    # YOUR CODE STARTS HERE
```

```
    step = 0
    V_old = np.ones(env.nS)

    while (np.linalg.norm(V_old-V) > theta and step < max_iteration):
        V_old = V                                                   # store old␣
↪state-value
        V = np.zeros(env.nS)                                        # initial␣
↪state-value to store new state-value
        policy = utils.extract_policy(env, V_old, gamma)            # use stored old␣
↪state-value to update policy

        # use updated policy to evaluate policy
        for state in range(env.nS):
            for nextS in env.P[state][policy[state]]:
                    V[state] = V[state] + nextS[0] * (nextS[2] + gamma *␣
↪V_old[nextS[1]])

        step += 1

    # YOUR CODE ENDS HERE
    ############################

    return V, policy
```

After implementing the above function, read and understand the functions implemented in evaluation_utils.py, which we will use to evaluate our value iteration policy

[12]:
```
from helpers import evaluation_utils
import gym
GAME = "Taxi-v3"
env = gym.make(GAME)
V_vi, policy_vi = value_iteration(env, gamma=0.95, max_iteration=6000,␣
 ↪theta=1e-5)
# visualize how the agent performs with the policy generated from value␣
 ↪iteration
evaluation_utils.render_episode(env, policy_vi)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : |_: |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| :_| : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R:_| : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
```

```
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 7.000000
```

[13]:
```
# evaluate the performance of value iteration over 100 episodes
evaluation_utils.avg_performance(env, policy_vi)
```

[13]: 8.05050505050505

3.2 2.2 Model-based RL: policy iteration

In this part, you are supposed to implement policy iteration to solve the Taxi game problem.

[14]:
```
from helpers import utils
def policy_iteration(env, gamma, max_iteration, theta):
    """Implement Policy iteration algorithm.

    You should use the policy_evaluation and policy_improvement methods to
    implement this method.

    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    gamma: float
            Discount factor.
    max_iteration: int
            The maximum number of iterations to run before stopping.
```

```python
    theta: float
            Determines when value function has converged.
    Returns:
    ----------
    value function: np.ndarray
    policy: np.ndarray
    """

    V = np.zeros(env.nS)
    policy = np.zeros(env.nS, dtype=int)
    ############################
    # YOUR CODE STARTS HERE

    step = 0
    policy_stable = False

    while (step < max_iteration and not policy_stable):
        V = policy_evaluation(env, policy, gamma, theta)
        new_policy, policy_stable = policy_improvement(env, V, policy, gamma)
        policy = new_policy

    # YOUR CODE ENDS HERE
    ############################

    return V, policy


def policy_evaluation(env, policy, gamma, theta):
    """Evaluate the value function from a given policy.

    Parameters
    ----------
    env: OpenAI env.
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
 ↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
                    number of actions

    gamma: float
            Discount factor.
    policy: np.array
            The policy to evaluate. Maps states to actions.
    max_iteration: int
```

```
            The maximum number of iterations to run before stopping.
    theta: float
            Determines when value function has converged.
    Returns
    -------
    value function: np.ndarray
            The value function from the given policy.
    """
    V = np.zeros(env.nS)
    ############################
    # YOUR CODE STARTS HERE

    max_iteration = 6000
    step = 0
    V_old = np.ones(env.nS)

    while (np.linalg.norm(V_old - V) > theta and step < max_iteration):
        V_old = V  # store old state-value
        V = np.zeros(env.nS)  # initial state-value to store new state-value

        # use updated policy to evaluate policy
        for state in range(env.nS):
            for nextS in env.P[state][policy[state]]:
                V[state] = V[state] + nextS[0] * (nextS[2] + gamma *␣
↪V_old[nextS[1]])

        step += 1

    # YOUR CODE ENDS HERE
    ############################

    return V


def policy_improvement(env, value_from_policy, policy, gamma):
    """Given the value function from policy, improve the policy.

    Parameters
    ----------
    env: OpenAI env
            env.P: dictionary
                    the transition probabilities of the environment
                    P[state][action] is tuples with (probability, nextstate,␣
↪reward, terminal)
            env.nS: int
                    number of states
            env.nA: int
```

```
                    number of actions

    value_from_policy: np.ndarray
            The value calculated from the policy
    policy: np.array
            The previous policy.
    gamma: float
            Discount factor.

    Returns
    -------
    new policy: np.ndarray
            An array of integers. Each integer is the optimal action to take
            in that state according to the environment dynamics and the
            given value function.
    stable policy: bool
            True if the optimal policy is found, otherwise false
    """
    ############################
    # YOUR CODE STARTS HERE

    policy_stable = True

    new_policy = utils.extract_policy(env, value_from_policy, gamma)

    policy_stable = (np.linalg.norm(policy-new_policy) == 0)

    # YOUR CODE ENDS HERE
    ############################

    return new_policy, policy_stable
```

[15]:
```
## Testing out policy iteration policy for one episode
GAME = "Taxi-v3"
evaluation_utils.render_episode(env, policy_vi)
env = gym.make("Taxi-v3")
V_pi, policy_pi = policy_iteration(env, gamma=0.95, max_iteration=6000,␣
 ↪theta=1e-5)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(West)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(West)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(North)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(North)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(West)

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

(Pickup)

```
+---------+
|R: | : :G|
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| :_: : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : :_: : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : :_: |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : |_: |
|Y| : |B: |
+---------+
  (South)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 7.000000
```

[16]:
```
# visualize how the agent performs with the policy generated from policy␣
↪iteration
evaluation_utils.render_episode(env, policy_pi)
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
```

```
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 14.000000
```

[17]:
```python
# evaluate the performance of policy iteration over 100 episodes
print(evaluation_utils.avg_performance(env, policy_pi))
```

```
8.292929292929292
```

4 Part 3: Q-learning and SARSA

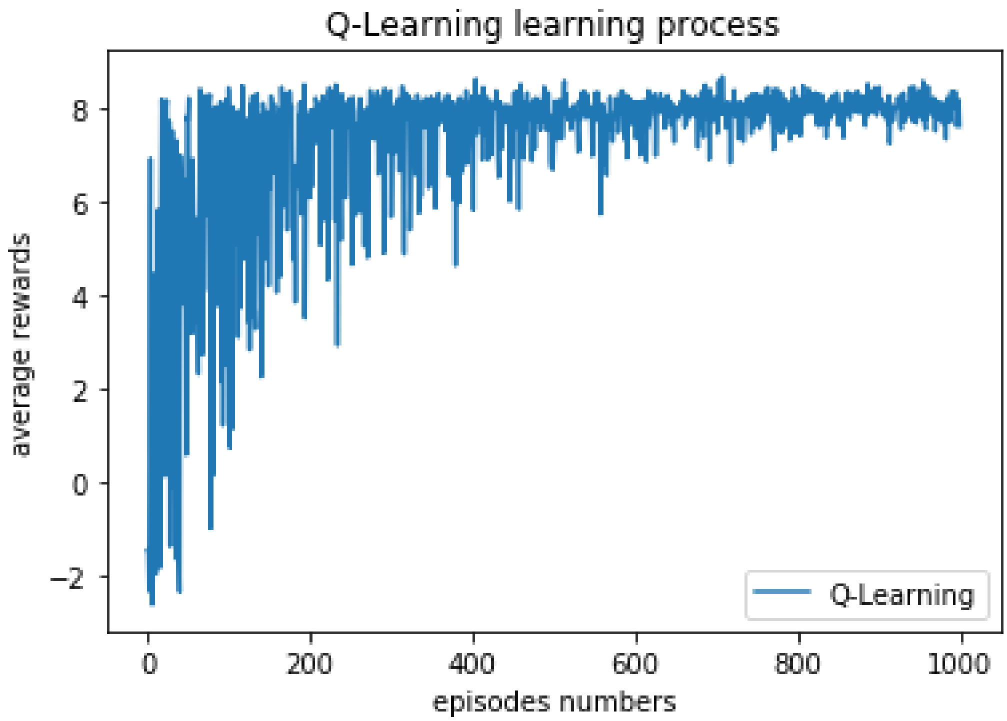
4.1 3.1 Model-free RL: Q-learning

In this part, you will implement Q-learning.

[18]:
```
def QLearning(env, num_episodes, gamma, lr, e):
    """
    Implement the Q-learning algorithm following the epsilon-greedy exploration.
    Inputs:
    env: OpenAI Gym environment
            env.P: dictionary
                    P[state][action] are tuples of tuples tuples with␣
 ↪(probability, nextstate, reward, terminal)
                    probability: float
                    nextstate: int
                    reward: float
                    terminal: boolean
            env.nS: int
                    number of states
            env.nA: int
                    number of actions
    num_episodes: int
            Number of episodes of training
    gamma: float
            Discount factor.
    lr: float
            Learning rate.
    e: float
            Epsilon value used in the epsilon-greedy method.
    Outputs:
    Q: numpy.ndarray
    """
    Q = np.zeros((env.nS, env.nA))

    ############################
    # YOUR CODE STARTS HERE

    max_iteration = 600
    average_reward_Q = [0 for i in range(num_episodes)]

    for episode in range(num_episodes):
        state = random.randint(0, (env.nS - 1)) # randomly initialize start␣
 ↪state
        step = 0

        while (not env.P[state][0][0][3] and step<max_iteration):
            step += 1
```

```
            action = epsilon_greedy(Q[state],e) # choose action from State␣
↪using e-greedy policy derived from Q
            reward = env.P[state][action][0][2]     # take action, get reward
            next_state = env.P[state][action][0][1]# take action, get next state
            diff = Q[next_state]
            Q[state][action] = Q[state][action] + lr * (reward + gamma *␣
↪Q[next_state][np.argmax(Q[next_state])] - Q[state][action])
            state = next_state
            average_reward_Q[episode] += reward


        if step != 0:
            average_reward_Q[episode] = average_reward_Q[episode] / step

    # Plot the learning process of both algorithms for training 1000 episodes.
    plt.plot([i for i in range(num_episodes)], average_reward_Q,␣
↪label="Q-Learning")
    plt.legend()
    plt.xlabel('episodes numbers')
    plt.ylabel('average rewards')
    plt.title('Q-Learning learning process')
    plt.show()

    # YOUR CODE ENDS HERE
    ############################

    return Q
```

[19]:
```
Q = QLearning(env = env.env, num_episodes = 1000, gamma = 1, lr = 0.1, e = 0.1)
print('Action values:')
print(Q)
```




```
Action values:
[[ 5.64252122e+04  5.66170106e+04  5.60858461e+04  5.65543375e+04
   5.71439526e+04  5.65559250e+04]
 [ 8.72557465e-01  3.82425687e+02 -2.00645819e+00 -2.08110690e+00
   8.64748642e+03 -2.93900311e+00]
 [ 9.29393408e+01 -3.00000000e-01  1.00589225e+01  2.00357163e+02
   2.92726334e+04 -2.75047591e+00]
 …
 [-6.81000000e-01 -6.49049000e-01 -6.00000000e-01 -6.33471000e-01
  -1.00000000e+00 -1.00000000e+00]
 [-1.00000000e+00 -9.60715159e-01 -1.00000000e+00  5.38863512e+03
  -1.96000000e+00 -2.78800000e+00]
 [ 4.01552706e+04  2.14996100e+04  3.59345833e+04  6.02492417e+04
   2.77840136e+04  2.80056848e+04]]
```

[20]:
```
# Uncomment the following to evaluate your result, comment them when you␣
 ↪generate the pdf
env = gym.make('Taxi-v3')
policy_estimate = utils.action_selection(Q)
evaluation_utils.render_episode(env, policy_estimate)
```

```
+---------+
|R: | : :G|
| : | : : |
```

```
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
```

```
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 12.000000
```

4.2 3.2 Model-free RL: SARSA

In this part, you will implement Sarsa.

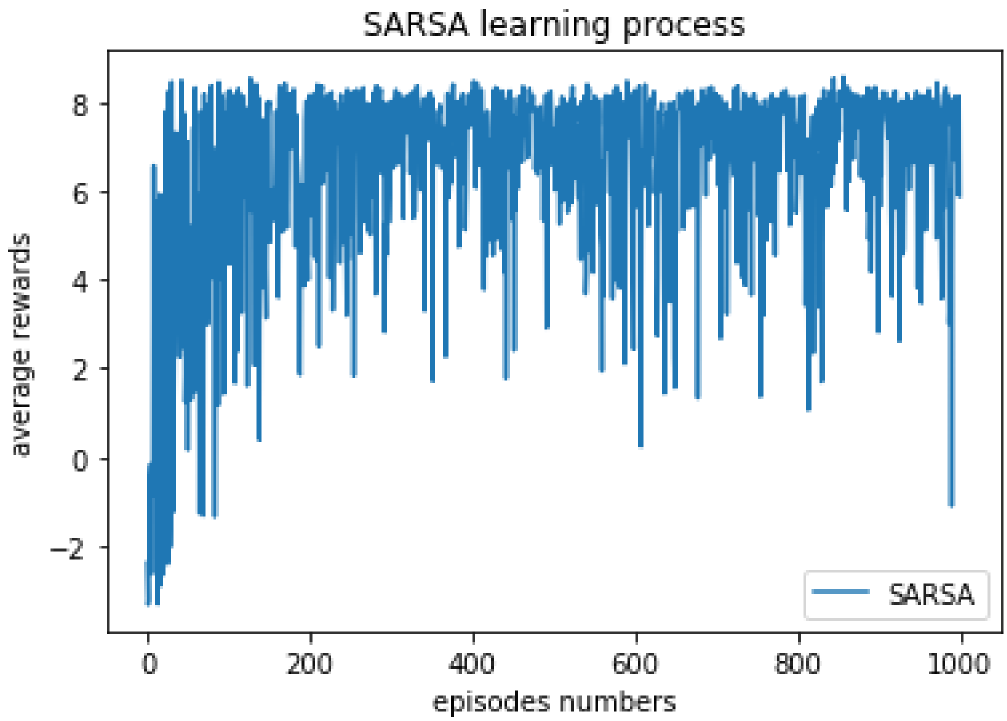
[21]:
```python
def SARSA(env, num_episodes, gamma, lr, e):
    """
    Implement the SARSA algorithm following epsilon-greedy exploration.
    Inputs:
    env: OpenAI Gym environment
            env.P: dictionary
                    P[state][action] are tuples of tuples tuples with␣
 ↪(probability, nextstate, reward, terminal)
                    probability: float
                    nextstate: int
                    reward: float
                    terminal: boolean
            env.nS: int
```

```
                    number of states
            env.nA: int
                    number of actions
    num_episodes: int
            Number of episodes of training
    gamma: float
            Discount factor.
    lr: float
            Learning rate.
    e: float
            Epsilon value used in the epsilon-greedy method.
    Outputs:
    Q: numpy.ndarray
            State-action values
    """
    Q = np.zeros((env.nS, env.nA))
    ############################
    # YOUR CODE STARTS HERE

    max_iteration = 600
    average_reward_SARSA = [0 for i in range(num_episodes)]

    for episode in range(num_episodes):
        state = random.randint(0, (env.nS - 1)) # randomly initialize start␣
↪state
        action = epsilon_greedy(Q[state],e) # choose action from State using␣
↪e-greedy policy derived from Q
        step = 0

        while (not env.P[state][0][0][3] and step<max_iteration):
            step += 1
            reward = env.P[state][action][0][2]     # take action, get reward
            next_state = env.P[state][action][0][1]# take action, get next state
            next_action = epsilon_greedy(Q[next_state],e)
            diff = [Q[next_state][a]-Q[state][action] for a in range(env.nA)]
            Q[state][action] = Q[state][action] + lr * (reward + gamma *␣
↪Q[next_state][next_action] - Q[state][action])
            state = next_state
            action = next_action
            average_reward_SARSA[episode] += reward

        if step != 0:
            average_reward_SARSA[episode] = average_reward_SARSA[episode] / step

    # Plot the learning process of both algorithms for training 1000 episodes.
    plt.plot([i for i in range(num_episodes)], average_reward_SARSA,␣
↪label="SARSA")
```

```
    plt.legend()
    plt.xlabel('episodes numbers')
    plt.ylabel('average rewards')
    plt.title('SARSA learning process')
    plt.show()

    # YOUR CODE ENDS HERE
    ############################

    return Q
```

[22]:
```
def render_episode_Q(env, Q):
    """Renders one episode for Q functionon environment.

      Parameters
      ----------
      env: gym.core.Environment
        Environment to play Q function on.
      Q: np.array of shape [env.nS x env.nA]
        state-action values.
    """

    episode_reward = 0
    state = env.reset()
    done = False
    while not done:
        env.render()
        time.sleep(0.5)
        action = np.argmax(Q[state])
        state, reward, done, _ = env.step(action)
        episode_reward += reward

    print ("Episode reward: %f" %episode_reward)
```

[23]:
```
Q = SARSA(env = env.env, num_episodes = 1000, gamma = 1, lr = 0.1, e = 0.1)
print('Action values:')
print(Q)
```




```
Action values:
[[ 3.33754612e+03  3.42232957e+03  3.39119039e+03  3.41182330e+03
   3.48026725e+03  3.42763159e+03]
 [ 3.06946343e+01  2.33804487e+01 -2.44530455e+00 -1.11064413e+00
   8.24403417e+02  1.44983312e+00]
 [-9.03406610e-01  6.08847208e+01 -7.51927273e-01 -7.67024785e-01
   1.50946519e+03 -2.73990000e+00]
 …
 [-6.52990000e-01  1.94057226e+02 -7.06063000e-01 -7.21245651e-01
  -1.90000000e+00 -1.91000000e+00]
 [-1.43251328e+00 -1.38881375e+00 -1.54481029e+00  5.96270634e+01
  -3.60753302e+00 -1.91000000e+00]
 [ 1.77856480e+03  2.06747057e+03  1.63308244e+03  3.49764855e+03
   1.99831891e+03  2.04284995e+03]]
```

[24]:
```
# Uncomment the following to evaluate your result, comment them when you␣
↪generate the pdf
env = gym.make('Taxi-v3')
policy_estimate = utils.action_selection(Q)
evaluation_utils.render_episode(env, policy_estimate)
```

```
+---------+
|R: | : :G|
| : | : : |
```

```
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : |_: |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
```

```
| : : : : |
| | : | :_|
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : | : : |
| : : : :_|
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : :_|
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
Episode reward: 11.000000
```

[ ]: