Exercise 7 Phil Szalay, Florian Schneider

January 30, 2020

```
In [70]: import numpy as np
         import random
         import matplotlib.pyplot as plt
         from statistics import mean
```

```
In [4]: number_of_possible_actions = 10
```

```
In [168]: def run_experiment(epsilon):
              # q: array of true rewards for action a
              q = np.random.normal(0, 1, number_of_possible_actions)
              optimal_action = np.argmax(q)

              number_of_optimal_actions = np.zeros(1000)

              # rewards: array of arrays of rewards for action a for every iteration
              rewards = np.zeros((2000, 1000))

              for j in range(2000):
                  # Q: array of estimated reward for action a
                  # n: dictionary of number of action a was choosen
                  Q = np.zeros(number_of_possible_actions)
                  n = np.zeros(number_of_possible_actions)

                  for i in range(1000):
                      random_number = random.uniform(0, 1)

                      # calc next action a
                      if (random_number > epsilon):
                          action = get_max_index(Q)
                      else:
                          random_index = random.randint(0, 9)
                          action = random_index

                      if (action == optimal_action):
                          number_of_optimal_actions[i] += 1

                      reward = bandit(action, q)
```

```
            rewards[j, i] = reward

            n[action] += 1
            Q[action] = Q[action] + 1 / n[action] * (reward - Q[action])

    # calc average Q for all actions
    reward_averages = np.mean(rewards, axis=0)
    number_of_optimal_actions_averages = number_of_optimal_actions / 2000

    return (reward_averages, number_of_optimal_actions_averages)

# returns draw from normal distribution with mean of Q[action]
def bandit(action, q):
    return np.random.normal(q[action], 1)

# returns index of max element, breaking ties randomly
def get_max_index(Q):
    return np.random.choice(np.flatnonzero(Q == Q.max()))
```

In [169]:

```
# epsilon = 0.1
a = run_experiment(0.1)

# epsilon 0.01
b = run_experiment(0.01)

# epsilon = 0
c = run_experiment(0)
```

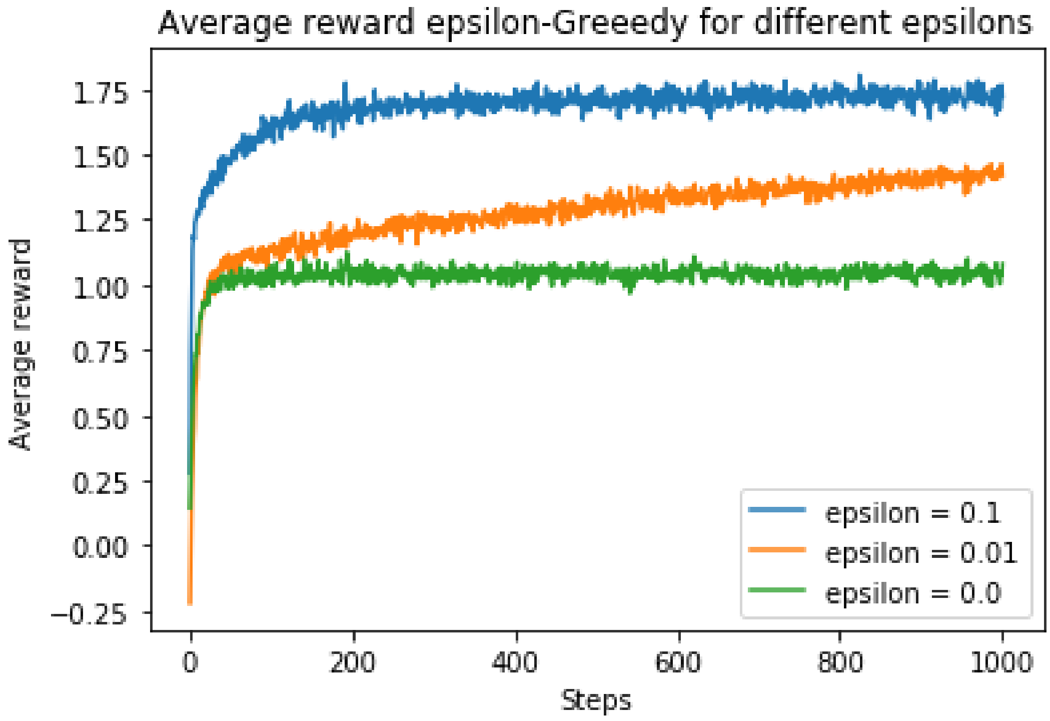
In [170]:

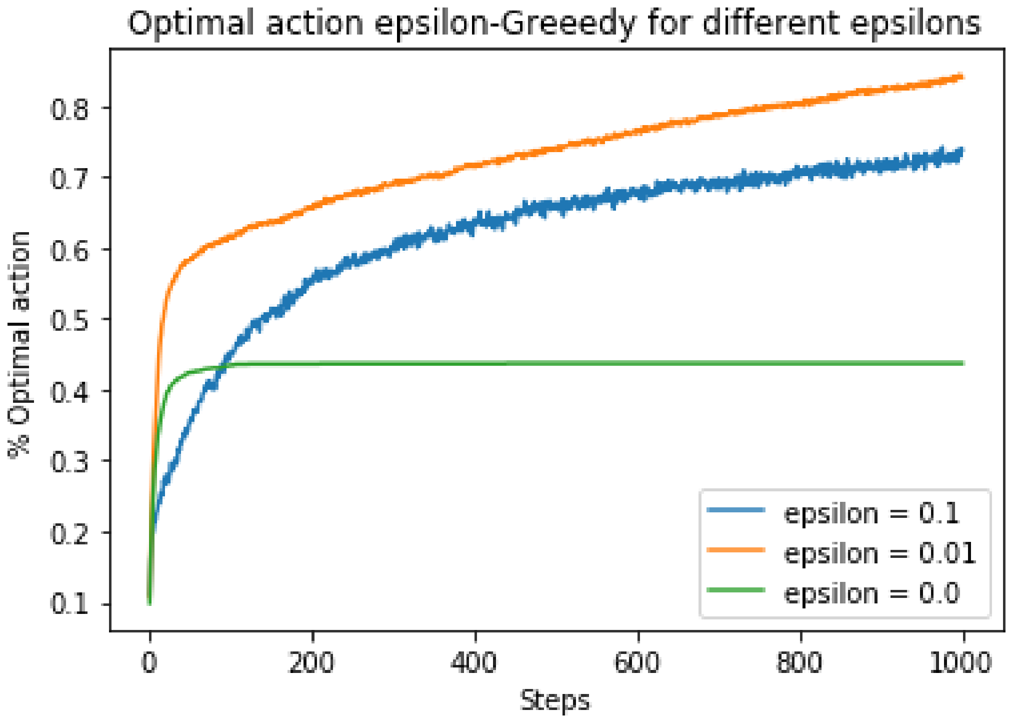
```
# plot all three curves
plt.plot(np.arange(1,  1001), a[0], label='epsilon = 0.1')
plt.plot(np.arange(1,  1001), b[0], label='epsilon = 0.01')
plt.plot(np.arange(1,  1001), c[0], label='epsilon = 0.0')
plt.legend()

plt.xlabel('Steps')
plt.ylabel('Average reward')
plt.title('Average reward epsilon-Greeedy for different epsilons')
plt.show()

# plot optimal choices
plt.plot(np.arange(1,  1001), a[1], label='epsilon = 0.1')
plt.plot(np.arange(1,  1001), b[1], label='epsilon = 0.01')
plt.plot(np.arange(1,  1001), c[1], label='epsilon = 0.0')
plt.legend()

plt.xlabel('Steps')
plt.ylabel('% Optimal action')
plt.title('Optimal action epsilon-Greeedy for different epsilons')
plt.show()
```


Average reward epsilon-Greeedy for different epsilons
Average reward
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
200
400
600
800
1000
Steps
epsilon = 0.1
epsilon = 0.01
epsilon = 0.0


Optimal action epsilon-Greeedy for different epsilons
% Optimal action
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
200
400
600
800
1000
Steps
epsilon = 0.1
epsilon = 0.01
epsilon = 0.0