Excercise 1

Kai Schneider

April 25, 2021

Task 1

a.)

$k = 2$, $\varepsilon = 0.5$
$\rightarrow P(\text{greedy}) = 1 - \varepsilon + \frac{\varepsilon}{k} = 1 - 0.5 + \frac{0.5}{2} = 0.75$
$\rightarrow P(\text{non-greedy}) = \frac{\varepsilon}{k} = \frac{0.5}{2} = 0.25$

b.)

$k = 4 \; \rightarrow a_i$ with $i = 1:4$, $\; Q_1(a_i) = 0$

with $A_t = \underset{a}{\text{argmax}} Q_t(a)$ as the greedy policy and $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_{i,a_i=a}}{n(a)}$ and the given data:

$$\begin{aligned} A_1 &= 1 & R_1 &= 1 \\ A_2 &= 2 & R_2 &= 1 \\ A_3 &= 2 & R_3 &= 2 \\ A_4 &= 2 & R_4 &= 2 \\ A_5 &= 3 & R_5 &= 0 \end{aligned}$$

I:

Step 1 (from Q_1 to Q_2) was definitely a random step because $Q_1(a_i) = 0 \, \forall i$, therefore the selection was arbitrary.

	a_1	a_2	a_3	a_4	action
Q_1	0	0	0	0	$A_1 = 1$
Q_2	1	0	0	0	$A_2 = 2$
Q_3	1	1	0	0	$A_3 = 2$
Q_4	1	3	0	0	$A_4 = 2$
Q_5	1	5	0	0	$A_5 = 3$
Q_6	1	5	0	0	-

A random selection also has to be occured in step 2 ($Q_2 \rightarrow Q_3$), because $A_2 = 2$ despite the $\underset{a}{\text{argmax}}$ being 1. Also in the fifth step ($Q_5 \rightarrow Q_6$) action A_3 was selected, which had to be a random selection too.

II:

In general a random step could have occured at any other point too. Especially step 3 ($Q_3 \rightarrow Q_4$) is a likely candidate, because the *argmax* is either 1 or 2. But even if the chosen A_i is the $\underset{a}{\operatorname{argmax}} Q_t(a)$, it is still possible that this was a random selection.

Task 2

c.)

As we can see that the ε-greedy policy works better in the long term than the greedy strategy. A reason might be that the greedy policy can get stuck at local maxima, while the exploring behaviour of the ε-greedy still allows for testing new/other machines.

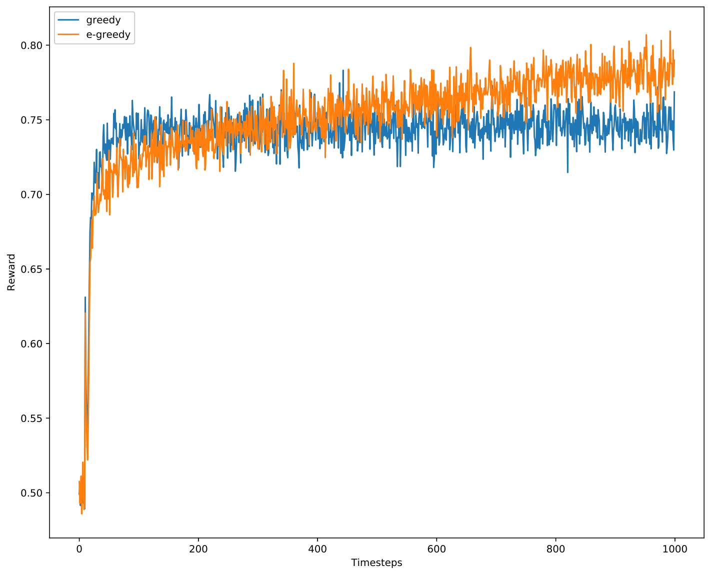

Figure 1: plot of the greedy (blue) and the ε-greedy (orange) policy

d.)

It might be useful to be more explorative in the beginning of our runtime to while being more exploitive the nearer we get to the end of our trials. This way we are more likely to find the most rewarding action(s) at the start and maximize our total reward towards the end.
This stratety would be especially useful for cases were we don't have that many trials.
For example we can start with a higher $\varepsilon = 0.5$ than before and slowly converge to our previous value $\varepsilon = 0.1$:

```
        eps = 0.5
        eps_add = float(0.4/timesteps)

        while bandit.total_played < timesteps:
            if random.random() < (eps - bandit.total_played*eps_add):
                a = random.randint(0,(bandit.n_arms-1))
            else:
                a = np.argmax(Q)
```

As you can see in in figure 2 this leads to less rewards in the beginning, but performes better than the previous methods after some time.

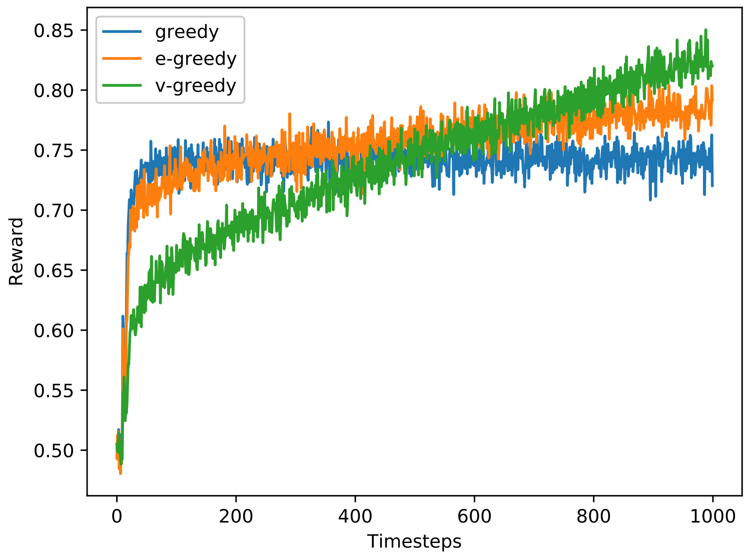

Figure 2: plot of the strategy with decreasing explorativity (green) compared with the previous methods