Uppsala University

Introduction to Machine Learning, Big Data, and AI

Assignment 9

General information

- The recommended tool in this course is R (with the IDE R-Studio). You can download R **here** and R-Studio **here**. You are allowed to use Python and Jupyter Notebooks, although the assignments may use data available only through the R package, a problem you would need to solve yourself.
- Report all results in a single, `*.pdf`-file. *Other formats, such as Word, Rmd, Jupyter Notebook, or similar, will automatically be failed.* Although, you are allowed to first knit your document to Word and then print the assignment as a PDF from Word if you find it difficult to get TeX to work.
- The report should be submitted to the **Studium**.
- To pass the assignments, *all questions should be answered*, although minor errors are ok.
- A report that do not contain the general information (see template) will be automatically rejected.
- When working with R, we recommend writing the reports using R markdown and the provided **R markdown template**. The template includes the formatting instructions and how to include code and figures.
- If you have a problem with creating a PDF file directly from R markdown, start by creating an HTML file, and then just print the HTML to a PDF.
- Instead of R markdown, you can use other software to make the pdf report, but the same instructions for formatting should be used. These instructions are also available in **the PDF produced from the R markdown template**.
- The course has its own R package `uuml` with data and functionality to simplify coding. To install the package just run the following:
  1. `install.packages("remotes")`
  2. `remotes::install_github("MansMeg/IntroML",`
     `subdir = "rpackage")`
- We collect common questions regarding installation, and technical problems in a course Frequently Asked Questions (FAQ). This can be found **here**.
- Deadline for all assignments is **Sunday at 23.59**. See the course page for dates.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue, or submit a pull request to the public repository!

1 Introduction to Reinforcement Learning

In this assignment, we will look into the basics of Reinforcement Learning. The Suggested reading for the following work is Sutton and Barto (2017), Chapter 2 and 3.

1.1 Bandits

In the first part, we will study the k-armed bandit problems more in detail.

1. Implement a stationary 5-armed bandit reward function that generates a reward R for a given category according to a true underlying (stationary) reward. Implement a function `stationary_bandit(a)` that takes an action $\mathcal{A} = (1, ..., 5)$. The reward function should return a reward according to
$$R \sim N(q^\star_a, 1)\,,$$
where $q^\star = (1.62, 1.20, 0.70, 0.72, 2.03)$. Below is an example how it *could* work. Note that since the function return a random value you might get a different result and it might still be a correct implementation.

```
set.seed(4711)
stationary_bandit(2)

## [1] 3.019735

stationary_bandit(1)

## [1] 2.99044
```

2. Also, we will also implement a non-stationary reward function. For all actions $\mathcal{A} = (2, ..., 5)$ the non-stationary should work as the stationary reward function. Although for action $a = 1$ the function should return a reward based on the iteration according to
$$R \sim N(-3 + 0.01t, 1)\,.$$
Implement the reward function as `nonstationary_bandit(a)` and it should work as follows. Again, note that these are random values, you might have another (correct) solution giving different results.

```
set.seed(4711)
nonstationary_bandit(2, t = 1)

## [1] 3.019735

nonstationary_bandit(1, t = 1)

## [1] -1.61956

nonstationary_bandit(1, t = 1000)

## [1] 8.196318
```

3. Implement a greedy policy algorithm that always chooses the greedy action, according to Eq. (2.2). The function called `greedy_algorithm(Q1, bandit)` should input a vector of initial estimates of the value of the five actions and a reward function, then run the bandit for 1000 steps and return the mean reward over all the 1000 steps, the value estimates $Q_t(a)$ and the total number of choices of each action $N_t(a)$. Here is an example of how it could work.

```
Q1 <- rep(0, 5)
set.seed(4711)
greedy_algorithm(Q1, bandit_stationary)
## $Qt
## [1] 1.593045 0.000000 0.000000 0.000000 0.000000
##
## $Nt
## [1] 1000    0    0    0    0
##
## $R_bar
## [1] 1.593045
```

4. Similarly, implement the ϵ-bandit algorithm that takes an additional argument ϵ that is the probability that the algorithm instead takes makes an explorative action. Implement this algorithm as `epsilon_algorithm(Q1, bandit, epsilon)`, note that it should return the same output structure as the greedy algorithm.

```
set.seed(4711)
Q1 <- rep(0, 5)
epsilon_algorithm(Q1, bandit_stationary, 0.1)
## $Qt
## [1] 1.6093183 1.1491153 0.7382012 0.5975228 2.0272784
##
## $Nt
## [1] 234  17  19  16 714
##
## $R_bar
## [1] 1.867178
```

5. Now implement the non-stationary bandit algorithm in a similar way with parameter α as `nonstationary_algorithm(Q1, bandit, epsilon, alpha)`.

```
set.seed(4711)
Q1 <- rep(0, 5)
nonstationary_algorithm(Q1, bandit_stationary, epsilon = 0.1, alpha = 0.2)
## $Qt
## [1] 1.414101 1.293336 1.237085 0.517214 2.134716
##
## $Nt
## [1] 302  16  19  16 647
##
## $R_bar
## [1] 1.840128
```

6. As a final step, also implement the UCB algorithm as a function called `ucb_algorithm(Q1, bandit, c)`. Hint! To solve numerical errors, you can use $\log(1.1)$ for $t = 1$ instead of $\log(t)$. Why will this not have any effect on the results?

```r
set.seed(4711)
Q1 <- rep(0, 5)
ucb_algorithm(Q1, bandit_stationary, c = 2)
## $Qt
## [1]  1.43350715  1.09734192 -0.01287212  0.64762674  2.02954004
##
## $Nt
## [1]  48  26   6  13 907
##
## $R_bar
## [1] 1.946474
```

7. Run each algorithm 100 times and compute the mean of the mean rewards for each of the algorithm. Do this for both the stationary and non-stationary bandit. For the non-stationary algorithm, try $\alpha = 0.5$ and $\alpha = 0.9$ and for the UCB, try $c = 0.5$ and $c = 2$. Summarize all the results in a table. Also, try optimistic initialization.
8. What are your conclusions from these results? Which algorithm seems to be the best one, and why?
9. For the best and worse algorithm, plot the average reward over 100 runs, such as in Figure 2.2. in Sutton and Barto (2017).

1.2 Markov Decision Processes

Now we are going to implement a Markov Decision Process (MDP) where the agent makes a decision, A_t based on the current state S_t and a reward R_t that will, in turn, return a new state S_{t+1} and a reward R_{t+1}. In this part of the assignment, we are going to use the recycling robot example (Example 3.7 in Sutton and Barto, 2017).

1. The MDP as in Sutton and Barto (2017) has been implemented as a function `recycling_mdp(alpha, beta, r_wait, r_search)` in the `uuml` R package and can be accessed as follows.

```r
library(uuml)
recycling_mdp(0.5, 0.8, 0.1, 1)

##       s s_next        a p_s_next r_s_a_s_next
## 1  high   high   search      0.5          1.0
## 2  high    low   search      0.5          1.0
## 3   low   high   search      0.2         -3.0
## 4   low    low   search      0.8          1.0
## 5  high   high     wait      1.0          0.1
## 6  high    low     wait      0.0          0.1
## 7   low   high     wait      0.0          0.1
## 8   low    low     wait      1.0          0.1
## 9   low   high recharge      1.0          0.0
## 10  low    low recharge      0.0          0.0
```

2. Now implement a function `always_search_policy(mdp)` that takes an MPD specified above and run the MDP for 1000 steps/actions. The policy should be that the agent always chooses to search, irrespective of the state. The function should return the return divided with the number of time steps and the number of times in each state. The robot should start in the state `high`. Below is an example of how it should work. Note again, since this is a random algorithm, you might get a slightly different answer.

```r
set.seed(4711)
always_search_policy(mdp)
## $Nt
## high  low
##  288  712
##
## $R_bar
## [1] 0.452
```

3. Now implement a function `charge_when_low_policy(mdp)` that takes an MPD specified above and run the MDP for 1000 steps. The policy should be that the agent always chooses to recharge if the energy level is low and always search if the energy level is high. The function should return the same thing as above.

4. Compare the two policies for MDP with $\alpha = \beta = 0.9$ and $\alpha = \beta = 0.4$ with $r_{\text{wait}} = 0.1$ and $r_{\text{search}} = 1$. Run at least ten times per policy and compute the mean reward. What policy is prefered in the two MDPs?

5. Compute the optimal value function for the states for the two different MDP above with a discount of $\gamma = 0.5$. What is the optimal value for the two different states? Hint, see Example 3.11. in Sutton and Barto (2017).

6. Implement a policy function called `optimal_greed(mdp)` that uses a *greedy* policy using the optimal values. Run the algorithm at least ten times for 1000 steps and compare with the two previous policies. What are your conclusions?