732A96/TDDE15 ADVANCED MACHINE LEARNING

EXAM 2021-01-05

Teacher

Jose M. Peña. Available by Teams **from 8.00 to 9.00 and from 10.00 to 11.00** (or by e-mail if Teams does not work).

Grades

- For 732A96 (A-E means pass):
  - A=19-20 points
  - B=17-18 points
  - C=14-16 points
  - D=12-13 points
  - E=10-11 points
  - F=0-9 points
- For TDDE15 (3-5 means pass):
  - 5=18-20 points
  - 4=14-17 points
  - 3=10-13 points
  - U=0-9 points

The total number of points is rounded to the nearest integer. In each question, full points requires clear and well motivated answers and/or commented code.

Instructions

This is an individual exam. No help from others is allowed. No communication with others regarding the exam is allowed. Answers to the exam questions may be sent to Urkund.

The answers to the exam should be submitted in a single PDF file using LISAM. You can make a PDF from LibreOffice (similar to Microsoft Word). You can also use Markdown from RStudio. Include important code needed to grade the exam (inline or at the end of the PDF file). The exam is anonymous, i.e. do not write your name anywhere.

Allowed help

Everything in the course webpage. Your individual and group solutions to the lab.

1. Graphical Models (6 p)

- (5 p) In Lecture 4, you learned about the PC algorithm. This is a structure learning algorithm based on independence hypothesis tests. You are asked to explain how the algorithm works on the `lizards` dataset, which comes with the `bnlearn` package. The **only** `R` function that you are allowed to use is the `ci.test` function, which runs an independence hypothesis test. Specifically, use the `x2` test, which was explained in Lecture 4. You can use any significance level that you deem appropriate, e.g. 0.05. You can read about the `lizards` dataset and see how to use the `ci.test` function here: `https://www.bnlearn.com/documentation/man/lizards.html`.

  Note that you are not asked to implement the PC algorithm. You just have to explain how it works on a concrete dataset.

- (1 p) Do you think that the PC algorithm learned the correct structure in the previous example ? Motivate your answer.

2. Hidden Markov Models (7 p)

- (5 p) In Lab 2, you used a HMM to model the behavior of a robot moving in a ring divided into 10 sectors. Recall that a HMM is nothing more than a dynamic Bayesian network, i.e. the nodes represent the same random variables at different time points. So, you are now asked to implement again the robot in Lab 2 but this time you are **only** allowed to use the `bnlearn` and `gRain` packages, i.e. no HMM specific package can be used. It suffices that you build the model for time 0 to time 3, i.e. four time slices.

- (2 p) Suppose that you received the following information from the **sensors**: The robot was in sector 1 in time 0 and in sector 3 in time 2. Where was really the robot (in distribution) in time 0, 1 and 2 ? Where is it going to be in time 3 ? Use the `bnlearn` and/or `gRain` packages to answer these questions.

3. Reinforcement Learning (7 p)

- (6 p) In this exercise, you are asked to consider a modified version of the environment C in Lab 4. The modified version differs from the original in that the reward for positions (1,2:5) is now -10 instead of -1, the reward for position (1,6) is still +10, and the reward for the rest of the positions is -1. **An episode ends now when the agent receives a reward of +10 or -10**, i.e. the episode does not end when the agent receives a reward of -1. The rest of the environment is the same as the original environment C.

  The Q-learning algorithm is based on the following updating rule:

  $$q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma \max_{a'} q(s',a') - q(s,a))$$

  which implies updating $q(s,a)$ to make it closer to $r+\gamma \max_{a'} q(s',a')$. Some may object that this does not make sense since it assumes that the agent acts greedily in the next state s' due to $\max_{a'}$, but in reality the agent acts ϵ-greedily. Then, they may argue that it would be better to use the following updating rule:

  $$q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma q(s',a') - q(s,a))$$

  which implies updating $q(s,a)$ as a function of the next state s' and action a'. In other words, one has to produce the next state and action in order to update the current state and action. This algorithm is called Sarsa or, more specifically, 1-step Sarsa. Recall that you were asked to implement it for the october 2020 exam. See `https://www.ida.liu.se/~732A96/exam/index.en.shtml`. You can also read more about it in Section 6.6 of the book by Sutton and Barto. This algorithm can be extended to 2-step Sarsa by using the following updating rule:

  $$q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma r' + \gamma^2 q(s'',a'') - q(s,a))$$

  which implies updating $q(s,a)$ as a function of the next reward r', and next next state s'' and action a''. In other words, one has to produce the next state and action and the next next state and action in order to update the current state and action. You are asked to implement 2-step Sarsa for the environment described above. You can reuse the implementation of 1-step Sarsa available at `https://www.ida.liu.se/~732A96/exam/index.en.shtml`. Page 147 of the book by Sutton and Barto contains pseudocode for n-step Sarsa, which may also help you. However, remember that you are just asked to implement 2-step Sarsa. **Comment your code**. Pay special attention to the case when s' or s'' are terminal states (i.e., the episode ends in those states, i.e. r or r' are -10 or +10).

- (1 p) Run 2-step Sarsa in the environment described above. Use $\epsilon = 0.5$, $\gamma = 1$, $\beta = 0$ and $\alpha = 0.1$. Run the algorithm for 5000 episodes and report the final q-table and policy.