EN.530.641: Statistical Learning for Engineers
Final Project

Jin Seob Kim, Ph.D.
Senior Lecturer, ME Dept., LCSR, JHU

out on 12/03/2021; due on 12/16/2021 by midnight EST

Instruction: You may use the textbooks, your class notes, handouts, and homeworks. You may also use internet resources (Scikit-Learn manual and so on), in which case you have to cite them properly. You can discuss the general concepts and aspects with the students in the class. However, no collaboration with other students (other than your team members) or help from people outside the course is allowed.

---

1 Task 1 : Machine Learning

In this task, you will apply three different algorithms that you learned in the class to either regression or classification problem. The procedure will be the following.

1. Your team will choose the dataset of interest (you have to consult your choice to TAs and the instructor). Then you will perform data preprocessing if necessary. Recall one of homework where you performed *Integer Coding* and *One-Hot Encoding*. That way, you can quantify the categorical output. Other data preprocessing techniques include:
    - Rescale the data: This is useful when features are very different ranges. In `sklearn`, you can use the MinMaxScaler class (i.e., `from sklearn.preprocessing import MinMaxScaler`).
    - Binarize the data: transform the data using a binary number (0 or 1). In `sklearn`, you can use Binarizer (i.e., `from sklearn.preprocessing import Binarizer`).
    - Standardize the data: transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. In `sklearn`, you can use StandardScaler (i.e., `from sklearn.preprocessing import StandardScaler`).

2. Perform the regression or classification depending on your choice. Apply three algorithms among:

   - Linear regression, or polynomial regression
   - Logistic regression, or linear discriminant analysis (LDA)
   - Artificial neural network
   - Support vector machine
   - Random forest

   Please be aware that in general you want to include regularization terms if possible (you know why).

3. For each of three algorithms you choose, perform model selection. In other words, for each model, select the best fit model by cross-validation (or out-of-bag error for random forest).

4. Now, you have three best models. Among them, select the best model by comparing test error (or generalization error).

In doing this task, you will need to divide the whole dataset. Use 90% of data for the training and validation, and use 10% to report the test error (or generalization error).

2 Problem for DP

Before delving into a task of reinforcement learning, let's have a taste for DP. In this problem, we consider Example 4.1 in Ch. 4 (p. 76) in RL book (also Fig. 4.1 in Ch. 4 in RL book, p. 77). Solve this problem by using the Policy Iteration. Write your python code, and report the final value function and the optimal policy. Use the equiprobable random policy.

3 Task 2 : Reinforcement Learning

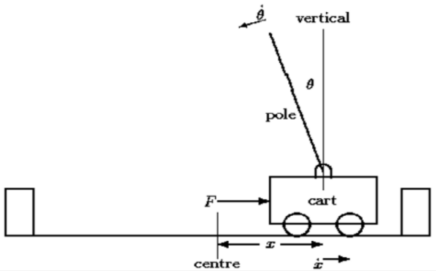

Figure 1: A schematic of the cart-pole system, adopted from [1].

In this task, you will apply reinforcement learning techniques to a 2-D inverted pendulum (cart pole system). See Fig. 1 for a schematic configuration. Specifically, you will apply two methods:

1. **Dynamic programming (DP)**: Try Policy Iteration and Value Iteration to compare. Note that *you do not use a model defined in* `CartPole-v0`*. i.e.,* `env.step()` *function. You have to implement the model (and the algorithms) in your python codes to perform the task..*

2. **Temporal difference (TD(0) or one-step TD) with Q-learning**: In this task, you can use OpenAI Gym, in particular, a model defined in `CartPole-v0` environment (i.e., use "learning from simulation" approach). Also use an ϵ-greedy policy.

3.1 The model for DP

Dynamic programming (DP) requires a model. For simplicity, let us consider the cart as a point (or particle) with mass M. Also the pole (pendulum) has the mass m with the half-pole length l. The model is written as the following nonlinear equations [2]

$$\begin{aligned}\ddot{\theta} &= \frac{g\sin\theta + \cos\theta\left[\frac{-F-ml\dot{\theta}^2\sin\theta+\mu_c\mathrm{sgn}(\dot{x})}{M+m}\right] - \frac{\mu_p\dot{\theta}}{ml}}{l\left[\frac{4}{3} - \frac{m\cos^2\theta}{m+M}\right]} \\ \ddot{x} &= \frac{F + ml\left[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c\mathrm{sgn}(\dot{x})}{M+m}\end{aligned} \tag{1}$$

where F is the force applied to the cart. The state of the system is defined as $s = (\theta, \dot{\theta}, x, \dot{x})$ (or in vector form as $\mathbf{s} = [\theta, \dot{\theta}, x, \dot{x}]^T$). The system equations can be numerically solved by the simple Euler's method as

$$\begin{aligned}\dot{x}(t+1) &= \dot{x}(t) + \Delta t\,\ddot{x}(t) \\ x(t+1) &= x(t) + \Delta t\,\dot{x}(t) \\ \dot{\theta}(t+1) &= \dot{\theta}(t) + \Delta t\,\ddot{\theta}(t) \\ \theta(t+1) &= \theta(t) + \Delta t\,\dot{\theta}(t)\end{aligned} \tag{2}$$

where t denotes the index of the discretized time. (1), (2) can be used to compute $p(s' \mid s, a)$ by using the Boxes system below. Note that you use tabular solution methods in this task.

3.2 OpenAI Gym

3.2.1 Introduction

The physical environment for the simulation can be obtained through "OpenAI Gym". Visit https://gym.openai.com to learn about it. The purpose of "OpenAI Gym" is to provide more standardized environments (enhance the repeatability of experiments). For the current task, you will want to choose `CartPole-v0`. To recap, for installation, you type in `pip install gym`, or use "conda" (`conda install -c conda-forge gym`) in the terminal window (in Mac system). After installation, as in the openai gym website, try the following to test:

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

The `make()` function creates an environment, and in this example, it is a cart-pole environment. Another important commands include `step()` function:

```
>>> obs, reward, done, info = env.step(action)
```

where outputs are observation (state), reward, and a boolean that determined the failure (end of an episode). You don't want to use `info` here. For more details, refer to the website https://gym.openai.com or [3].

3.2.2 An example for DP

The following code (from [3]) shows an example of DP by using OpenAI Gym:

```
import gym
env = gym.make('CartPole-v0')

def basic_policy(obs):
    angle = obs[2]
    return 0 if angle < 0 else 1

totals = []
for episode in range(500):
    episode_rewards = 0
    obs = env.reset()
    for step in range(1000): # 1000 steps max, we don't want to run forever
        action = basic_policy(obs)
        obs, reward, done, info = env.step(action)
        episode_rewards += reward
        if done:
            break

    totals.append(episode_rewards)

env.close()
```

This is a very simplified version (e.g., the policy is unique and fixed). But still you can copy and paste the code to run. This way, you will be able to have a sense of how gym is working.

Note that in DP task, *you have to implement the algorithms as learned in the class in your python codes to perform the task, not just using the model defined in Gym as in the above example.*

3.2.3 An example for Q-learning

As an example of Q-learning, please refer to the following example code [3]:

```
import numpy.random as rnd

learning_rate0 = 0.05
learning_rate_decay = 0.1
n_iterations = 20000
```

```
s = 0 # start in state 0

Q = np.full((3, 3), -np.inf) # -inf for impossible actions
for state, actions in enumerate(possible_actions):
    Q[state, actions] = 0.0 # Initial value = 0.0, for all possible actions

for iteration in range(n_iterations):
    a = rnd.choice(possible_actions[s]) # choose an action (randomly)
    sp = rnd.choice(range(3), p=T[s, a]) # pick next state using T[s, a]
    reward = R[s, a, sp]
    learning_rate = learning_rate0 / (1 + iteration * learning_rate_decay)
    Q[s, a] = learning_rate * Q[s, a] + (1 - learning_rate) * (
                reward + discount_rate * np.max(Q[sp])
            )
    s = sp # move to next state
```

This example is not specifically for the cart-pole problem. But it shows how to implement Q-learning.

3.3 Numerical details

For numerical details, we employ those in [2]. Let $M = 1.0$ kg, $m = 0.1$ kg, $g = -9.8$ m/s^2, $l = 0.5$ m, $\mu_c = 0.0005$, and $\mu_p = 0.000002$. The action a is in fact the force applied to the cart. Following [2], we assume $F = \pm 10$ N. In other words, $a \in \{10, -10\}$. Time interval for numerical integration (or discrete form of the state equations) is $\Delta t = 0.02$ seconds.

3.4 The Boxes system

In order to implement any RL algorithm, first you need to discretize (or quantize) the state set. Recall that RL algorithms in our class deal with discrete spaces of actions, states, and rewards. We will use the Boxes system [4, 2, 1]. Specifically, we will follow [2]. The boxes system for the state space is constructed by the following quantization thresholds:

1. θ : 0, ± 1, ± 6, $\pm 12°$;
2. x : ± 0.8, ± 2.4 m;
3. $\dot{\theta}$: ± 50, $\pm\infty°$/s;
4. $\dot{x}$: ± 0.5, $\pm\infty$ m/s.

This generates $6 \times 3 \times 3 \times 3 = 162$ regions through all combinations of the intervals. By employing the boxes, the system becomes discrete and tabular. Also see [2, 4, 1] for more details.

Note that you can modify the resolution of the boxes system as necessary.

3.5 Goal and Reward

The goal of this task is to balance the pole by moving the cart. Specifically, balance is considered: 1) θ measured from the vertical line remains $\pm 12°$; and 2) the cart position remains within ± 2.4 m from the centre of the track.

Specifically, there can be two scenarios following the textbook, [5]:

- If θ and x is within the range ($|\theta| \leq 12°$ and $|x| \leq 2.4$ m) at each step, then the reward is $+1$, while failure at a certain step renders the reward as 0 (end of an episode). In the class, we said that it is without discounting. But you can add γ here.
- Reward is -1 for the failure, and 0 otherwise, with discounting.

Try any of them (or you can use your own reward definition). Just provide a clear explanation of your choice.

3.6 Task Outcome

The outcome of DP should include:

- Plots of value function. Since there are 4 state variables, generate plots of value function with respect to θ and x for selected $\dot{\theta}$ and $\dot{x}$. At least three plots are needed.
- Plots of optimal policy, in the same way as in the plots of value function.

The outcome of TD should include:

- Plots of total reward received per episode (averaged over 10 trials) vs. number of episodes, with three different discount rates.
- Plot of the cumulative reward (the sum of all rewards received so far) as a function of the number of episodes, with three different discount rates.
- Plots of x and θ as functions of time for some successful episodes together with number of time steps during which the pole does not fail). (see http://gym.openai.com/docs/#building-from-source). Note that you can find more in the website.

Also it has been used that, from [6], the problem is "considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials." This can help you to have a sense of successful trials.

Deliverables

- Project report (in pdf format, maximum 6 pages) that contains detailed information on your models, resulting plots required as outcome, as well as proper reference list. Also the report should contain team workload distribution among team members.
- *All* python codes used in the project.

Project Submission Guideline

- There are two submission sites in the gradscope. First, your team must submit a single zip file that contains all python codes to "Final Project: Codes". Name the single zip file submission of your team as "Team#_NameInitials_FP.zip". For example, "Team1_SD_SC.zip" assuming that Siming and Samarth are team 1. Second, submit your team's report (in pdf format) to "Final Project: Report". Only one person in the team should submit the teamwork for the project.
- Please make sure to include *all the necessary files*. If TAs try to run your function and it does not run, then your submission will have a significant points deduction.
- Make as much comments as possible so that your codes are readable.

References

[1] S. Nagendra, N. Podila, R. Ugarakhod, and K. Geroge. Comparison of reinforcement learning algorithms applied to the cart-pole problem. *arXiv:1810.01940 [cs.LG]*, 2017.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'REILLY, 2017.

[4] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. *Machine Intelligence*, 2(2):137–152, 1968.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[6] https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py.