MSBD 6000M
Assignment 1 Report

LAU Ho Yin, 20825377

2025-03-17

Table of Contents

Assignment Question

Consider the discrete-time asset allocation example in section 8.4 of Rao and Jelvis. Suppose the single-time-step return of the risky asset as Y_t=a, prob = p, and b, prob = (1 - p). Suppose that T=10, use the TD method to find the Q function, and hence the optimal strategy.

Objective

Maximize expectation on the CARA Utility Function.

Approach & Assumptions

The Q-Learning algorithm has been implemented to find the optimal policy. The algorithm was implemented in Python, NumPy and Pandas.

In order to formulate the problem into an Markov Decision Process solvable with Q-Learning, the following assumptions have been made:

1. Instead of what is specified in section 8.4 of Rao and Jelvis, which states that the reward distribution after each time step follows a normal distribution, in this assignment the reward follows a Bernoulli distribution, where there can only be 2 possible rewards, `a` or `b`.
2. It was given in section 8.4 of Rao and Jelvis that the optimal action at time step does not depend on the Wealth accumulated thus far, and that the optimal policy for a fixed time `t` is a constant deterministic policy function. This means the state only depends on discrete time `t`.
3. While asset allocation is a continuous variable, the mapping into the MDP action space is discretized to simplify the implementation of the Q-Learning algorithm. It is noted that continuous action space can be implemented, but is beyond the scope of this project.

Source Code

Source code can be accessed at https://github.com/dennishylau/msbd6000m_a1.

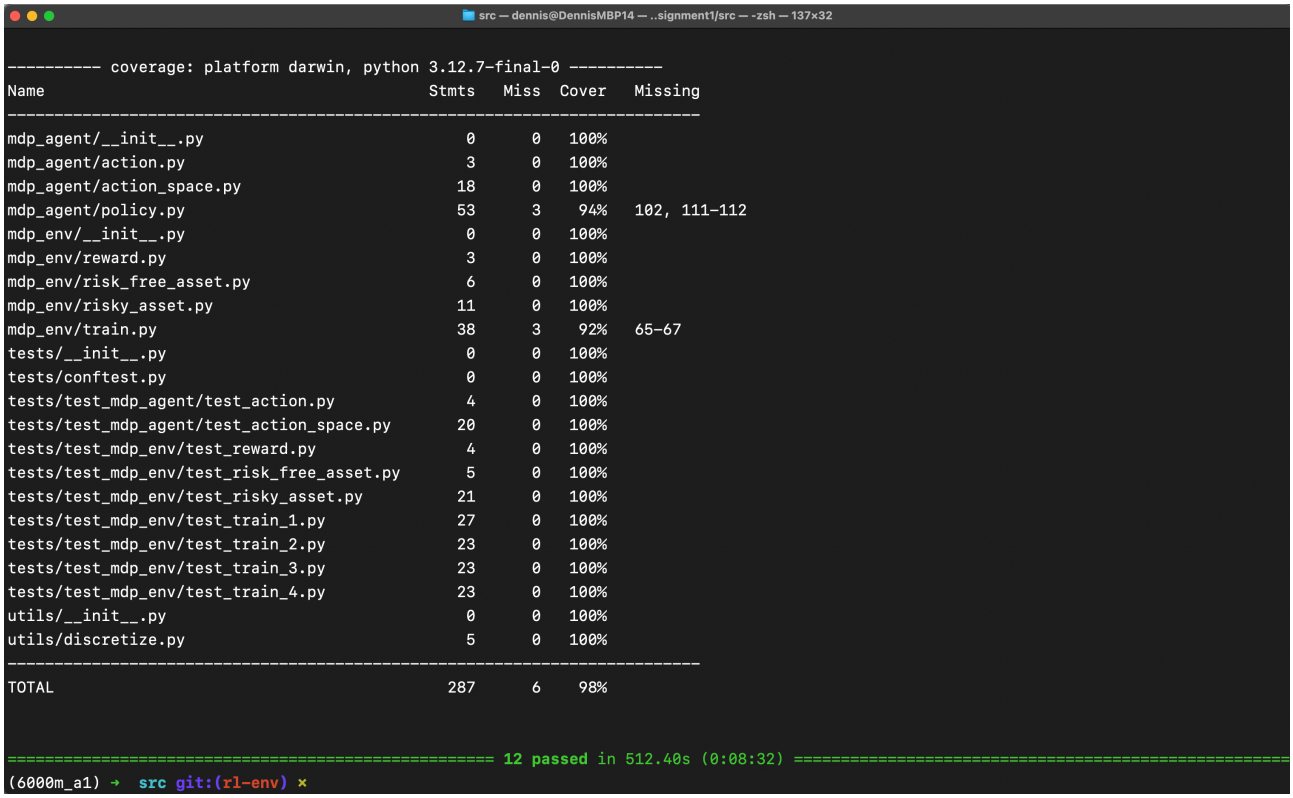
Execution of Code

Python source code is placed under the `src` folder. Within the folder, the `README.md` file has detail instructions on how the code can be executed.

For ease of visual evaluation, a `main.ipynb` Jupyter Notebook has been prepared to illustrate the training process for a list of risky asset scenarios, as well as the Q-Table and Optimal Policy for each scenario as an output of training. The process of convergence is also demonstrated in the Jupyter Notebook cells' output.

Code Quality

Within the `README.md` file, instructions on how to execute automated testing has also been included. Referencing Google's practice[1] of "exemplary" coverage of 90%, this project has a code coverage of 98%.



For each of the scenarios covered in `main.ipynb`, the resultant policy from training is compared against the analytical solution given by $V_t^*(W_t) = max_\pi \left(E_\pi \left[\frac{-e^{-aW_T}}{a} \middle| (t, W_t) \right] \right)$ as part of test cases in `test_train`-prefixed Python files.

[1] https://testing.googleblog.com/2020/08/code-coverage-best-practices.html