CS7646 ML4T
Machine Learning
for Trading

PROJECT 1: MARTINGALE

REVISIONS

This assignment is subject to change up until 3 weeks prior to the due date. We do not anticipate changes; any changes will be logged in this section.

- 5/18/2021 Update 'Code' section of Rubric removing "if not" from "Does the code display charts in a window or screen? (-10 points each up to a max of -20 if not)".
- 5/21/2021 Update Experiment 1/2 instructions replacing "upper" with "plus" and "lower" with "minus" for further clarification of standard deviation plots.

OVERVIEW

In this project, you write software that will perform probabilistic experiments involving an American Roulette wheel. The project will help provide you some initial feel for risk, probability, and "betting." Purchasing a stock is, after all, a bet that the stock will increase in value. You will submit the code for the project Gradescope SUBMISSION. You will also submit to Canvas a report that discusses your experimental findings.

ABOUT THE PROJECT

Build a Simple Gambling Simulator

Revise the code in martingale.py file to simulate 1000 successive bets on spins of the American roulette wheel using the betting scheme outlined in the pseudo-code below. Each series of 1000 successive bets are called an "episode". You should test for the results of the betting events by making successive calls to the get_spin_result(win_prob) function. Note that you will have to update the win_prob parameter according to the correct

probability of winning. You can figure that out by thinking about how roulette works (see Wikipedia link below).

In this project, you will evaluate the actual betting strategy that Professor Balch uses at roulette when he goes to Las Vegas.

Here is the pseudocode of the strategy:

```
episode_winnings = $0
while episode_winnings < $80:
 won = False
 bet_amount = $1
 while not won
 wager bet_amount on black
 won = result of roulette wheel spin
 if won == True:
 episode_winnings = episode_winnings + bet_amount
 else:
 episode_winnings = episode_winnings - bet_amount
 bet_amount = bet_amount * 2
```

Additional details regarding how roulette betting works: Betting on black (or red) is considered an “even money” bet. That means that if you bet N chips and win, you keep your N chips and you win another N chips. If you bet N chips and you lose then those N chips are lost. The odds of winning or losing depend on whether you are betting at an American wheel or a European wheel. For this project, we will be assuming an American wheel. You can learn more about roulette and betting here: https://en.wikipedia.org/wiki/Roulette.

YOUR IMPLEMENTATION

To write your code, download the starter code (see below). After downloading the starter code, you will develop an implementation, leveraging the pseudocode above, that conducts several experiments. Conduct the following experiments, then write your report. Before the deadline, make sure to pre-validate your submission using Gradescope TESTING. Once you are satisfied with the results in testing, submit the code to Gradescope SUBMISSION. Only code submitted to Gradescope SUBMISSION will be graded.

Experiment 1 – Explore the strategy and make some charts

In this experiment, you will develop code that performs experiments using Professor Balch's original betting strategy. You will run some experiments to determine how well the betting strategy works. The approach we are going to take is called Monte Carlo simulation where the idea is to run a simulator repeatedly with randomized inputs and to assess the results in aggregate. Your implementation will produce the following charts (i.e., figures):

- Figure 1: Run your simple simulator 10 episodes and track the winnings, starting from 0 each time. Plot all 10 episodes on one chart using Matplotlib functions. The horizontal (X) axis must range from 0 to 300, the vertical (Y) axis must range from –256 to +100. Note that we will not be surprised if some of the plot lines are not visible because they exceed the vertical or horizontal scales.
- Figure 2: Run your simple simulator 1000 episodes. (Remember that 1000 successive bets are one episode.) Plot the mean value of winnings for each round of spin using the same axis bounds as Figure 1. For example, you should take the mean of the first spin of all 1000 episodes. Add an additional line above and below the mean, at mean plus standard deviation, and mean minus standard deviation of the winnings at each point.
- Figure 3: Use the same data you used for Figure 2 but plot the median instead of the mean. Add an additional line above and below the median to represent the median plus standard deviation and median minus standard deviation of the winnings at each point.

For all of the above figures and experiments, if the target $80 winnings are reached, stop betting and allow the $80 value to persist from spin to spin (e.g., fill the data forward with a value of $80).

The charts created by the experiments will be included in your report, along with your supporting analysis and discussion. All charts must be properly titled, have appropriate axis labels, use consistent axis ranges, and have legends.

Experiment 2 – A more realistic gambling simulator

You may have noticed that the original strategy performed in experiment 1 works well, maybe better than you expected. One reason for this is that we were allowing the gambler to use an unlimited bankroll. In this experiment, we are going to make things more realistic by giving the gambler a $256 bankroll. This will require a modification to the original strategy since if he or she runs out of money: bzzt, that's it. Repeat the experiments, as above, with this new condition. Note that once the player has lost all their money (i.e., episode_winnings reach -256) stop betting and fill that number (-256) forward.

An important corner case to be sure you handle is the situation where the next bet should be $N, but you only have $M (where M<N). Make sure you only bet $M. Here are the two charts to create:

- Figure 4: Run your realistic simulator 1000 episodes and track the winnings, starting from 0 each time. Plot the mean value of winnings for each spin using the same axis bounds as Figure 1. Add an additional line above and below the mean at mean plus standard deviation and mean minus standard deviation of the winnings at each point.
- Figure 5: Use the same data you used for Figure 4 but plot the median instead of the mean. Add an additional line above and below the median to represent the median plus standard deviation and median minus standard deviation of the winnings at each point.

The charts created by the experiments will be included in your report, along with your supporting analysis and discussion. All charts must be properly titled, have appropriate axis labels, use consistent axis ranges, and have legends. You should use python's Matplotlib library.

TECHNICAL REQUIREMENTS

The martingale.py file must implement this API specification.

All winnings must be tracked by storing them in a NumPy array. You might call that array winnings where winnings[0] should be set to 0 (just before the first spin). The entry in winnings[1] should reflect the total winnings after the first spin and so on.

Use the population standard deviation. The standard deviation is plotted as two lines, the upper standard deviation (+stdev) and the lower standard deviation (–stdev).

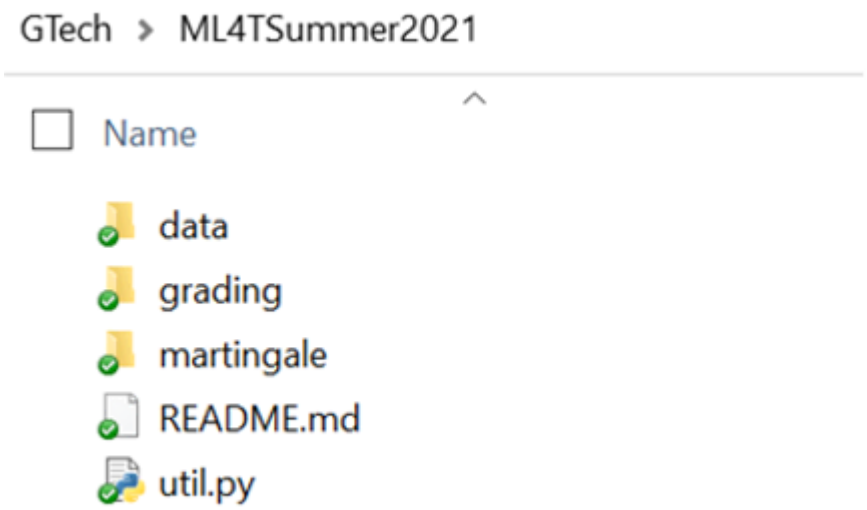
You may set a specific random seed for this assignment. If a specific random seed is used, it can only be called once, and it must use your GT ID as the numeric value.

Textual output may be "printed" to the terminal/console.

Starter Code

To make it easier to get started on the project and focus on the concepts involved, you will be given a starter framework. This framework assumes you have already set up the local environment and ML4T Software. The framework for Project 1 can be obtained from: Martingale_2021Summer.zip.

Extract its contents into the base directory (e.g., ML4T_2021Summer). This will add a new folder called "martingale" to the directory structure.



Within the martingale folder is a single file: martingale.py. You will modify the martingale.py file to implement the necessary functionality for this assignment. The existing code in the martingale.py file may contain ideas for functions and methods that might be used in your implementations. This file must remain and run from within the martingale directory using the following command:

```
PYTHONPATH=../:. python martingale.py
```

CONTENTS OF REPORT (100 POINTS)

In addition to submitting your agent to Gradescope, you will also write up a report describing your experimental hypothesis, design, and findings. While the analysis can be supported with theoretical or mathematical proof, the assignment requires the production and evaluation of empirical results (i.e., results based on experimental observation). Your submitted project should include all the code necessary to generate the charts presented in your report. **Up to -30 points in deductions will be applied to the report score for unmet implementation requirements or code that fails to run.**

Your report must use the JDF format, which specifies font sizes and margins that should not be altered. Charts must be generated by the code (including any desired annotations) and saved as .png files. They can be imported (without additional post-processing or editing) into the report document. Charts must be properly annotated with legible and appropriately named labels, titles, and legends. All charts must use the same axis bounds.

Answer the following prompt in a maximum of 7 pages (excluding references) in JDF format. Any content beyond 7 pages will not be considered for a grade. The analysis and

responses must be supported by experimental evidence:

> **Question 1**: In Experiment 1, based on the experiment results *calculate and provide the estimated probability of winning $80 within 1000 sequential bets*. *Thoroughly explain your reasoning for the answer using the experiment output*. Your explanation should NOT be based on estimates from visually inspecting your plots.
>
> **Question 2**: In Experiment 1, *what is the estimated expected value of winnings after 1000 sequential bets? Thoroughly explain your reasoning for the answer*.
>
> **Question 3**: In Experiment 1, *do the mean upper standard deviation line and mean lower standard deviation lines reach a maximum (or minimum) value and then stabilize? Do the standard deviation lines converge as the number of sequential bets increases? Thoroughly explain why it does or does not*.
>
> **Question 4**: In Experiment 2, based on the experiment results *calculate and provide the estimated probability of winning $80 within 1000 sequential bets*. *Thoroughly explain your reasoning for the answer using the experiment output.* Your explanation should NOT be based on estimates from visually inspecting your plots.
>
> **Question 5**: In Experiment 2, *what is the estimated expected value of our winnings after 1000 sequential bets? Thoroughly explain your reasoning for the answer*.
>
> **Question 6**: In Experiment 2, *do the mean upper standard deviation line and mean lower standard deviation lines reach a maximum (or minimum) value and then stabilize? Do the standard deviation lines converge as the number of sequential bets increases? Thoroughly explain why it does or does not.*
>
> **Question 7**: *What are some of the benefits of using expected values when conducting experiments instead of simply using the result of one specific random episode?*

Hint: If the upper and lower standard deviation lines do not converge and/or stabilize, explain why they do not. If they converge and/or stabilize, also discuss the value(s) at which they do so.

TESTING

A separate local testing script is not provided for this assignment. You are encouraged to perform any tests necessary to instill confidence that the code will run properly when submitted for grading and will produce the required results. To run and test that the file

will run from within the martingale directory, use the command given above at the command prompt from within the martingale directory:

```
PYTHONPATH=../:. python martingale.py
```

Note: Once submitted for grading, we will use the above command to call the "__main__" section only. The program should run in its entirety and produce the necessary output and charts.

You are encouraged to submit your files to Gradescope TESTING, where some basic pre-validation tests will be performed against the code. Gradescope TESTING does not grade your assignment. No credit will be given for coding assignments that does not pass this pre-validation.

You are allowed **unlimited** resubmissions to Gradescope **TESTING**. Please refer to the Gradescope Instructions for more information.

SUBMISSION INSTRUCTIONS

This is an individual assignment. All work you submit should be your own. Make sure to cite any sources you reference and use quotes and in-line citations to mark any direct quotes.

Late work is not accepted without advanced agreement except in cases of medical or family emergencies. In the case of such an emergency, please contact the Dean of Students.

Report Submission

Complete your assignment using the JDF format, then save your submission as a PDF. The report is to be submitted as **report.pdf.** Assignments should be submitted to the corresponding assignment submission page in Canvas. You should submit a single PDF for this assignment. Please submit the following file to Canvas in PDF format only:

- **report.pdf**

Do not submit any other files. Charts must be included in the report, not submitted as separate files. Also note that when we run your submitted code, it should generate all five (5) figures as .png files.

You are allowed unlimited submissions of the report.pdf file to Canvas.

Code Submission

This class uses Gradescope, a server-side auto-grader, to evaluate your code submission. No credit will be given for code that does not run in this environment and students are encouraged to leverage Gradescope TESTING prior to submitting an assignment for grading.

Please submit the following file to Gradescope SUBMISSION:

- **martingale.py**

Do not submit any other files.

You are allowed a MAXIMUM of three (3) code submissions to Gradescope SUBMISSION.

GRADING INFORMATION

Your report is worth 100% of your grade. As such, it will be graded on a 100-point scale coinciding with a rubric design to mirror the questions above (see rubric below). Make sure to answer those questions. The submitted code is run as a batch job after the project deadline. All points for the assignment will be returned in the Canvas report score.

We do not provide an explicit set timeline for returning grades, except that everything will be graded before the institute deadline (end of the term). As will be the case throughout the term, the grading team will work as quickly as possible to provide project feedback and grades. Once grades are released, any grade-related matters must follow the Assignment Follow-Up guidelines and process.

RUBRIC

Report

- Are the questions answered correctly? (Up to -5 points for each incorrect answer)
- Is the reasoning for each question correct and supported by the evidence thoroughly? (Up to -5 points for each if incorrect)
- Is each of the charts provided and correct and include labeled axes and legend? (Up to -8 points for each if incorrect)

Code

- Does the code run without crashing? (-10 points if not)
- Does the code generate appropriate charts written to .png files? (-10 points each up to a max of -20 if not)
- Does the code display charts in a window or screen? (-10 points each up to a max of -20)
- Does the implemented code reflect the project requirements? (Up to -10 points if not)

Auto-Grader

- No auto-grader

Allowed

- All libraries provided on the Local Environment setup page.
- All standard Python libraries (except the os library, which is prohibited)
- Code provided by the instructor or allowed by the instructor to be shared.

Prohibited

- Use of the Python os library/module.
- Use of global variables.
- Code that results in a window or chart being displayed (an example of which is Matplotlib's plt.show() function).
- Code that creates any directories.
- Any code that you did not write yourself (except for properly cited code provided by the instructor).
- Absolute import statements, such as: *from folder_name import martingale*.
- Any libraries not included as part of the Local Environment or Software Setup pages.
- Editing (i.e., adding, changing, or deleting) anything (including spaces) above the line that reads: "—–do not edit anything above this line—".
- Sharing of Charts, tables, and statistics.
- Knights who say "neeee."

RELEVANT RESOURCES

You may find the following resources useful in completing the project or in providing an in-depth discussion of the material.

Wikipedia (accessed 2021), Expected Value

Martelli, A. Ravenscroft, and S. Holden (2017), Python in a Nutshell, 3rd Edition

James, D. Witten, T. Hastie, R. Tibshirani (2017), An Introduction to Statistical Learning (Chapter 2)

Murphy, (2021), Probabilistic Machine Learning: An Introduction (Chapter 2)