Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.008 Introduction to Inference
Fall 2021

Computational Lab I

Issued: Wednesday, September 8, 2021 **Due:** Friday, September 24, 2021

Python Version: Make sure to run your code using Python Version 3.6.9 or newer.

Optional Reading: A useful Python reference is, e.g., the Python Scientific Lecture Notes, at `https://scipy-lectures.github.io/`. The NumPy information in Section 1.4 and the matplotlib information in Section 1.5 are likely to be particularly helpful in getting started with the computational labs.

Investing

This lab is a good (and hopefully fun) introductory lab and Python warm-up exercise. You'll implement two investment strategies and make some observations of their properties, which you'll learn how to analyze as a simple application of the inference tools we develop later in the class.

Code and Data. In this computational lab, you are provided with the starter file `stocks.zip`. Download the file from the course website and extract it to your working directory. It contains

- Folder `data` with two csv files, `priceA.csv` and `priceB.csv`
- File `stocks.py`, which will serve as the starting template for your code. In particular, note the following two functions that you will need in your code for parts (b) and (e):
  - `compute_average_value_investments()` for part (b), which returns 2 float values
  - `compute_doubling_rate_investments()` for part (e), which returns 2 float values

  (More details are provided in the respective prompts.)

Important: Do not change the function definitions in the code as they must remain consistent for automatic grading. Do not import additional modules since these modules might not exist in the autograder environment. The only parts of the code that you need to fill in are in blocks denoted by "`YOUR CODE GOES HERE`". If you would like, feel free to create additional functions as needed to help with the computation or to answer the questions in the lab, although this is not required. See the comment at the beginning of each function you complete for what the expected input and output are. Also, you must be using Python 3; we recommend version 3.6.9 or above.

Suppose you have some money and need to decide what to do with it. You can put it under your mattress, in a bank account, or invest it in some combination of bonds, stocks, or other funds. To simplify the problem, we'll just consider two choices among these possibilities: Investment A and Investment B. This is our investment "portfolio." Assume that each day[1] an investment has a return—money invested at the start of that day increases or decreases in value by the end of that day, reaching a new value.

Among various investment strategies, two classic ones are the following.

> **Buy and Hold:** At the start of the first day, we split our initial wealth of into two portions, putting the first portion into Investment A and the second into Investment B. We then let the value of the resulting *buy and hold portfolio* evolve according to the returns of these investments over time (i.e., many days) without further intervention.
>
> **Constant Rebalancing:** At the start of the first day, we again invest our initial wealth according to the buy and hold strategy. But at the start of each subsequent day, we rebalance our portfolio as follows: we take the total value of our portfolio at the start of that day, and reallocate it between the two investments so the proportion matches that used at the start of the first day. So if, e.g., Investment A earns a better return than Investment B in a given day, then at the start of the next day we will sell some of Investment A and purchase some more of Investment B in order to adjust the proportion of our wealth in each to be the same as at the start. The result is referred to as a *constant rebalanced portfolio.*

The buy and hold strategy is obviously simplest. And the constant rebalancing strategy is at least a bit counterintuitive since we will generally sell some of the investment that is doing better every day. But which strategy is better? Let's investigate.

First, let's compare these two investment strategies on a pair of stocks corresponding to two large (real) companies: Stock A and Stock B. In the files `priceA.csv` and `priceB.csv` we've provided, you'll find the daily stock price history of the two companies from August 1995 to August 2015, corresponding to roughly 5000 days over the 20 year period.[2]

(a) Implement both strategies on this data, starting with a \$1.00 investment and partitioning this equally into the two stocks (i.e., \$0.50 into each). Plot the

[1] Investment periods need not be a single day, but for simplicity, let's restrict our attention to this case.

[2] The data are aligned by date—the first rows of each correspond to day one, second rows of each correspond to day two, etc.

performance. Specifically, in single figure plot the wealth (i.e., total value of the portfolio) in each case as a function of the number of days since the investing started. Describe what you observe, and offer an interpretation.

To further analyze and compare these two investment strategies, we'll now use simulated data for two different investments: Investment C and Investment D. Investment C is a fixed income investment, earning 5% interest every day, i.e., money invested grows by a factor $\gamma = 1.0500$ every day. Investment D is more volatile: on a given day, its value changes by a factor of either $\beta = 1.4000$ (i.e., grows) or $\alpha = 0.7875$ (i.e., shrinks).

For an investment window of n days, there are then 2^n possible patterns of returns for Investment D.[3] For each of these possible patterns, you will compare the buy and hold and constant rebalancing strategies. Let $n = 20$, and as in part (a), start with a \$1.00 investment and partitioning this equally into Investments C and D (i.e., \$0.50 into each).

To help you get started generating (and ultimately checking) your code, note that at the end of the first day, the \$0.50 invested in Investment C is worth $\$0.50 \cdot \gamma = \$0.525$, and the \$0.50 invested in Investment D is worth $\$0.50 \cdot \beta = \$0.70$ or $\$0.50 \cdot \alpha = \$0.39375$, depending on whether the investment grows or shrinks. Suppose that it grows. Then at the beginning of the second day, the total wealth is now $\$0.525 + \$0.70 = \$1.225$, so constant rebalancing will put $\$1.225/2 = \$0.6125$ into each of Investments C and D. Thus, at the end of the second day, the value of investment C in the constant rebalanced portfolio will be $\$0.6125 \cdot \gamma = \$0.643125$, and that of investment D will be $\$0.6125 \cdot \beta = \$0.8575$ or $\$0.6125 \cdot \alpha = \$0.48234375$, depending on whether Investment D grows or shrinks on the second day.

(b) Compute the average value of the portfolio at the end of the investment window for each of the two strategies, where the averaging is over all the possible return patterns for Investment D. Which performs better, buy and hold or constant rebalancing?

Write your code for this part in the `compute_average_value_investments` function in `stocks.py`, returning the average value of each of the two portfolios (`average_buyandhold, average_rebalancing`).
Hint: In writing your code, you may find the Python function `numpy.binary_repr()` useful.

(c) For each of the return patterns, evaluate which strategy yields the bigger portfolio value at the end of the investment window. For what fraction of the patterns does buy and hold perform better?

(d) How does the run-time of your code in part (c) grow as you increase n from $n = 1$ to $n = 20$? Generate a plot of the run-time (on a logarithmic scale) as a

[3] For example, for $n = 3$, the patterns would be $\alpha\alpha\alpha$, $\alpha\alpha\beta$, $\alpha\beta\alpha$, $\alpha\beta\beta$, $\beta\alpha\alpha$, $\beta\alpha\beta$, $\beta\beta\alpha$, and $\beta\beta\beta$.

function of n (on a linear scale). Roughly estimate how long it would take for your code to run if you were to try $n = 30$ (but do not run your code with this value of n).

When the value of an investment after n days is w_n, we refer to

$$r_n \triangleq \frac{1}{n} \log_2(w_n)$$

as the wealth *doubling rate* (or *growth exponent*). In this part, you will evaluate the doubling rates for the two strategies involving Investments C and D. But now, instead of considering all patterns of returns for Investment D, you will only consider patterns where there are an equal number of up and down days over the n day window, i.e., $n/2 = 10$ days yielding return β and $n/2 = 10$ days yielding return α.

(e) Compute the doubling rate for each of the buy and hold and constant rebalancing strategies above. Which is larger?

Write your code for this part in the `compute_doubling_rate_investments` function in `stocks.py`, returning the doubling rate of each of the two strategies (`doubling_rate_buyandhold, doubling_rate_rebalancing`).

(f) **(Optional, not graded—just for your enjoyment!)** Suppose in the buy and hold strategy, instead of putting half of your initial wealth of \$1.00 in Investment C, you more generally put a fraction $0 \leq \lambda \leq 1$. Is it possible to achieve a higher doubling rate by choosing $\lambda \neq 1/2$? Likewise, is it possible to achieve a higher doubling rate for the constant rebalancing strategy by choosing $\lambda \neq 1/2$?

What to upload: Upload your writeup as a PDF file and upload your code as `stocks.py`, to Gradescope. The writeup should include your answers and plots for parts (a)–(e). Scans/images of handwritten work are fine but please write neatly — We can't grade what we can't decipher! Plots should be part of the writeup and not uploaded as separate files.

Gradescope will run a number of tests which are visible to you. These are to make sure your code runs correctly in the Gradescope system. Please make sure you pass these tests.