University Of Georgia

Final Project Report

Extension of Project 4 – A Market Simulator

Demo: https://youtu.be/pwxmbb3Jwc8

Nathan Brooks

CSCI 4170

Prof. Hybinette

Dec. 12, 2022

In homework 4, our goal was to create a market simulator. In this homework assignment, we read in premade order books in csv format and used its entries to simulate buying and selling stocks with historical market data. This historical market data was stored in a collection of csv file exported from Yahoo Finance. I found this simulator to work very well. It made creating and scanning order books a very simple process that I felt could be turned into something much larger, so it was a no-brainer to try and extend it to include more functionality. What if I could create an algorithm to generate these order books for me? Is there a way to extend this further to incorporate machine learning? So that's exactly what I set out to do. I wanted to create a program where I could create, and back test different strategies based on historical data.

I wanted to design a system where I could load and test multiple strategies and use a variety of different indicators to create these strategies. I was originally inspired by Trading View's strategy back tester. This is on online software I've used in the past to view both live and historical market data. They include an awesome library of tools that anyone can use to program and test their own custom strategies. However, I wanted to take this concept and recreate it in python with the hopes of adding my own custom machine learning libraries.

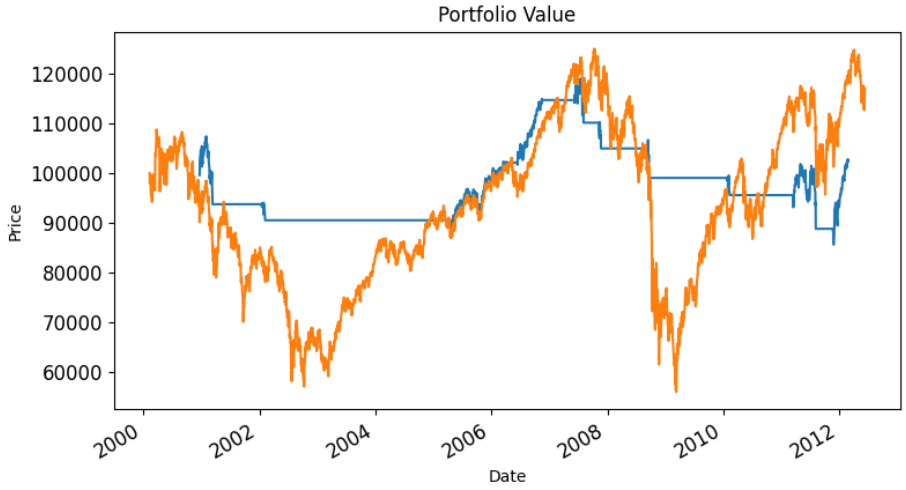
After plenty of research and much trial and error, I came up with a solution on the best way to store and test custom strategies. Since strategies are mostly built using indictors to alert investors on buy and sell signals, I first needed a way to calculate and store these indicators for use in my strategies while also allowing for variable input to allow for tweaking and editing the indicators' performance. I also wanted to easily graph and view these indicators next to market

data to easily visualize their performance. I accomplished this by creating an indicators directory where I could add functions representing each of the indicators. Here, I could pass in historical stock values as a Pandas data frame and append new columns to store the indicator metrics. As a default program base, I added two indicators: Bollinger Bands and Relative Strength Index (RSI). Bollinger bands returned metrics for the moving average as well as the high and low band while RSI included the RSI values for each candle in the stock dataset. Now I had an easy group of indicator functions that I could import into my strategies to calculate buy or sell signals.

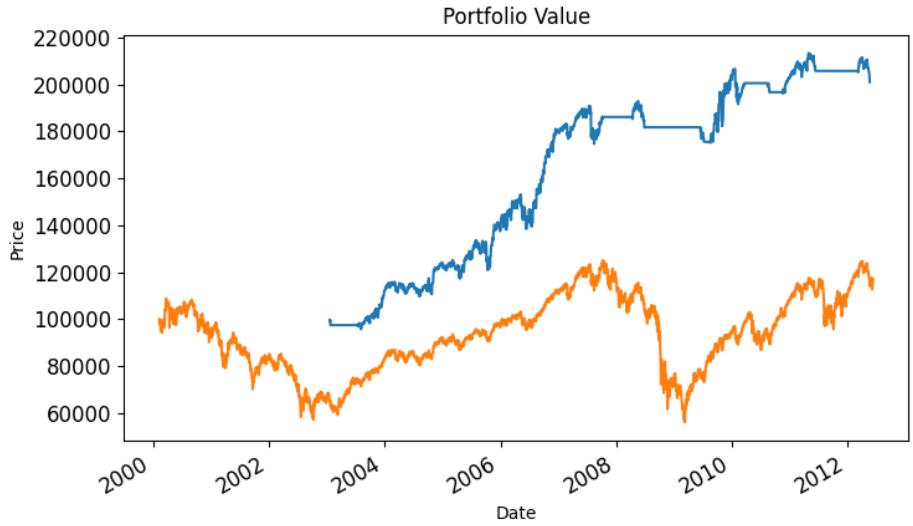
Next, I implemented strategy functionality. Since I would eventually be adding multiple strategies to my project, I wanted to create an easy way for myself and others to extend my project and add new strategies of their own. To do this, I created an abstract strategy class as a base that others could easily extend to create their own strategies around. This class allowed others to easily create and place both stop losses and take profit, include multiple open positions, and easily place buy or sell orders. Using this new class, I was able to create my first strategy.

I based my initial strategy on a common concept used by investors and popular banks called value averaging or VA. VA is an investment strategy which involves making regular contributions into a target security over time. With VA you invest more when the price of a security falls and less when it rises. I wanted to adapt this strategy for use in a custom strategy, but also implement stop losses and take profits. My goal was to time my market entries with technical indicators so that I could catch the market on a rise therefore triggering my stop loss before my take profit.

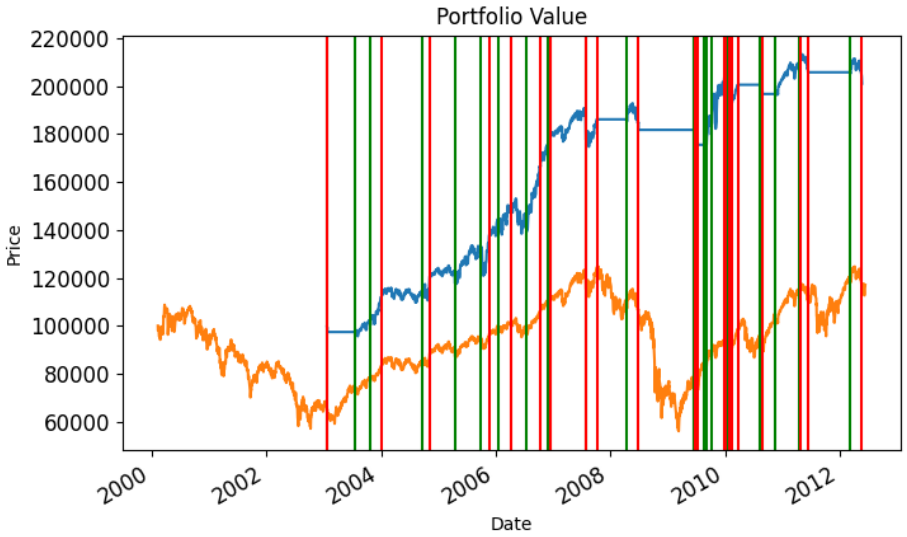
I used SPY for testing my strategy and tried to optimize most of the strategy parameters specifically for it as I thought it was the best representation of the overall market. I tested this specifically on the daily timeframe from 2000 all the way to 2012 and after much testing, I finally created a strategy that seemed to hold up well to market conditions. Below, you can see how my strategy performs using the default parameters:



This graph is using SPY as the baseline (orange) and you can see how my strategy performs in comparison (blue). This was a huge breakthrough for me. As you can see above, my algorithm managed to completely avoid the giant downturns in the market (2002-2005 and 2009-2010). At this point I was only using 1 open position at a time and only traded with $1000. I didn't beat the market, but I was able to make a positive return on investment. I further tweaked the strategy parameters a little bit and started getting some very impressive results for this time period.



Here, I slightly adjusted the Bollinger band's window size, the RSI threshold, and increased the max number of open positions to 3. Trading with just $3000, I was able to increase my portfolio over $100k during the 12-year time span. Below, I have included the same graph but with the green and red lines representing my buy and sell orders for visual clarity:



I quickly went to continue testing my strategy on different stocks and timeframes with varying results.

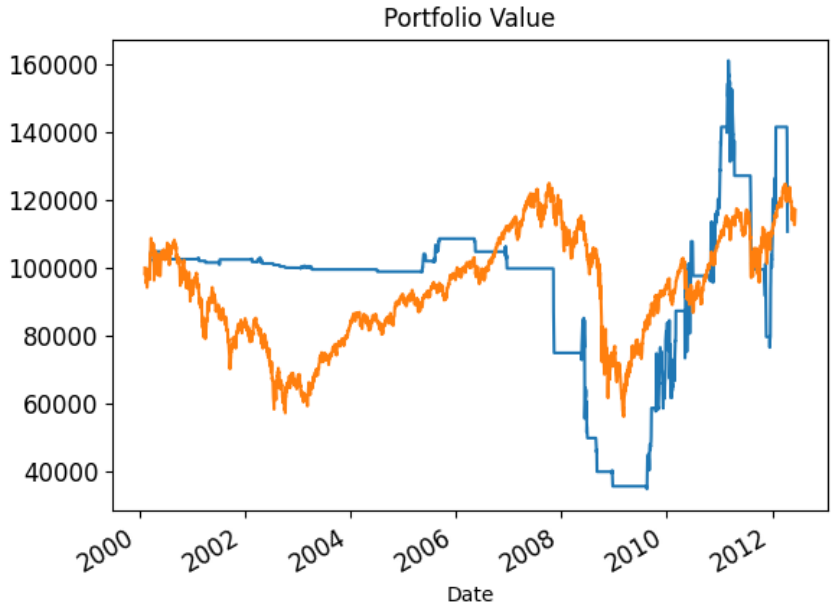


Figure 1: Ticker AAPL (2000-2012)

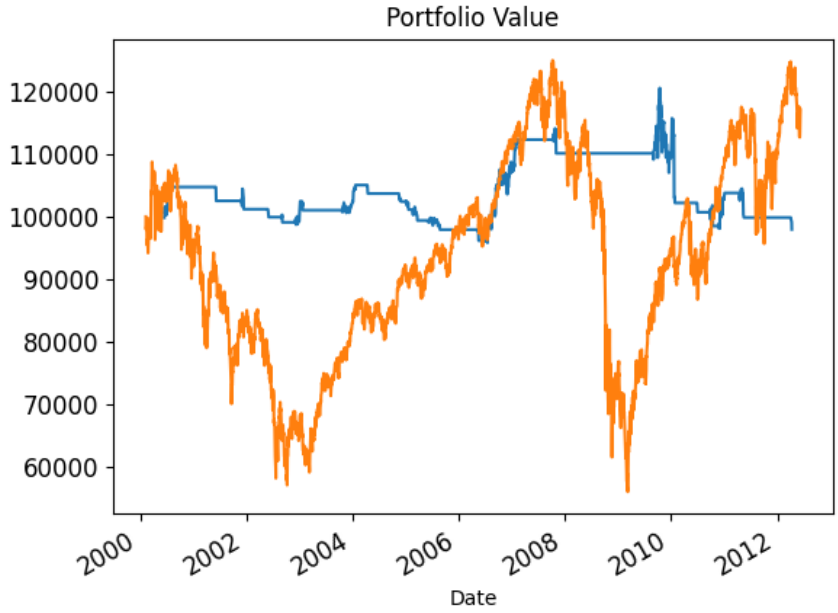


Figure 2: Ticker JPM (2000-2012)

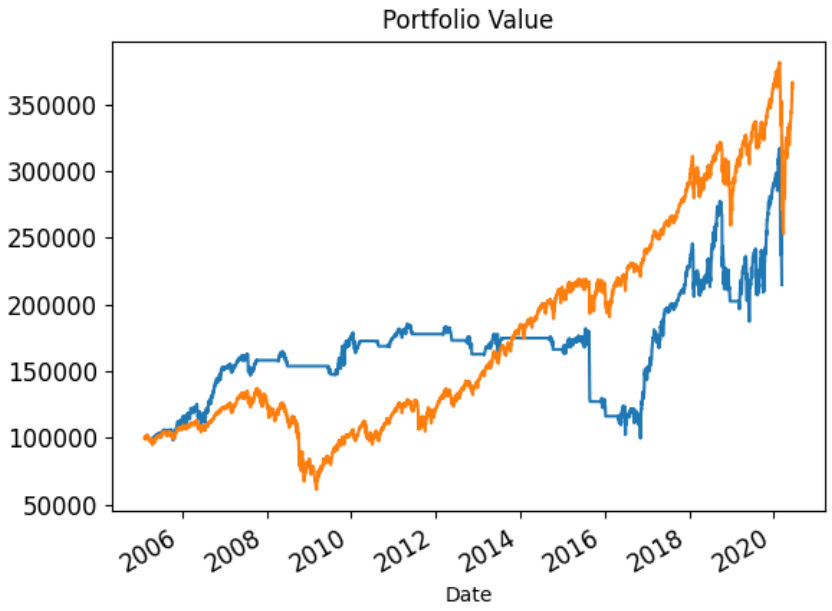


Figure 3: Ticker SPY (2006-2020)

Clearly, I hadn't found the perfect trading strategy since the results weren't optimal. But I was still impressed with the performance of the strategy and its ability to hedge well against market downturns. I feel that by optimizing the parameters for this strategy for specific stocks and timeframes It could deliver some promising results. This got me thinking and I realized that I wanted to further extend this simulator to implement machine learning.

My initial plan to implement machine learning was to use an optimizer like we did in class for Homework 5. I would use SciPy to optimize the strategy parameters to maximize profit on specific time periods and stocks. However, during implementation of this optimizer, I quickly learned that it was a much different problem than I originally anticipated. After much research, I realized that since my strategy parameters used integers it would not be compatible with any of SciPy's optimization solvers. Because my parameters were both integers and floating-point numbers, it made my problem a non-linear, mixed-integer optimization problem. This would

not work with SciPy since it's meant to work on non-integer continuous functions only. Even if I used a different library to optimize for profit, because of the state of the problem, it could not be guaranteed to be solved optimally unless I tried every possible combination. For example, my indicator uses 7 different parameters with a wide range of different inputs. Combined, these parameters result in about (20 * 20 * 100 * 100 * 10 * 50 * 50 = 100,000,000,000) 100 billion combinations, which is not feasible to solve. But maybe there's another type of machine learning algorithm better suited to solve a problem like this. Further research suggests that I could use reinforcement learning to reach a similar result.

In the future, I would really like to explore different ways of using machine learning to optimize and improve my strategies and create an investment powerhouse. I know that my strategies can be improved a lot from what they are now since there is an infinite number of different ones out there using all types of indicators and signals. I'm excited about the progress I made and am looking forward to testing new theories. Hopefully I can even find a way to implement machine learning to further test and improve too.