AUTOMATE STRATEGIES FOR MULTI STOCK TRADING

SUBJECT: MACHINE LEARNING IN FINANCE (MATH 527)

By:

Nandan Pandey - A20493184
Raj Shah - A20524266

INTRODUCTION

Stock trading is usually considered as one of the most challenging applications due to volatility in the market and several other reasons. Many researchers have explored various approaches using Deep Reinforcement Learning. Volatility scaling can be incorporated with DRL to trade futures contracts. By adding a market volatility term to reward functions, we can scale up the trade shares with low volatility, and vice versa. News headline sentiments and knowledge graphs can also be combined with the time series stock data to train an optimal policy using DRL.
High frequency trading with DRL is also a hot topic. Deep Hedging represents hedging strategies with neural networks learned by modern DRL policy search.
This application has shown two key advantages of the DRL approach in quantitative finance, which are scalability and model independent. It uses DRL to manage the risk of liquid derivatives, which indicates further extension of the FinRL library into other asset classes and topics.

. PROBLEM STATEMENT

To design a reinforcement learning based financial model that helps in deciding trading strategies for multiple stock trading in order to obtain maximum portfolio value based on current and the current position ($v_{t =} c_t + x_0 p_t$) using attributes such as open, close, high, low and other Technical indicators such as date, tic, operating margin, net profit margin, return over assets, return on equity, earnings per share, book per share etc.
Here, v_t is total portfolio value, c_t is the current available cash and $x_0 p_t$ represents current position. Our aim is to maximize the total portfolio value v_t by selecting the efficient trading strategies.

LIBRARY USED

Here, we are going to use a three-layered FinRL library that streamlines the development of stock trading strategies.
FinRL provides common building blocks that allow strategy builders to configure stock market datasets as virtual environments, to train deep neural networks as trading agents,
to analyze trading performance via extensive backtesting, and to incorporate important market frictions. On the lowest level is environment, which simulates the financial market environment using actual historical data from six major indices with various environment attributes such as closing price, shares, trading volume, technical indicators etc.

In the middle is the agent layer that provides fine-tuned standard DRL algorithms (DQN, DDPG, Adaptive DDPG, Multi-Agent DDPG, PPO, SAC, A2C and TD3 , etc.), commonly used reward functions and standard evaluation baselines to alleviate the debugging workloads and promote the reproducibility. The agent interacts with the environment through properly defined reward functions on the state space and action space.

Upper layer consists of Applications such as Stock Trading Portfolio Allocation, High Frequency Trading etc,

ENVIRONMENT

TIME-DRIVEN TRADING SIMULATOR

Considering the stochastic and interactive nature of the automated stock trading tasks, a financial task is modeled as a Markov Decision Process (MDP) problem. The training process involves observing stock price change, taking an action and reward's calculation to have the agent adjusting its strategy accordingly. By interacting with the environment, the trading agent will derive a trading strategy with the maximized rewards over the time. Our trading environments, based on OpenAI Gym framework, simulate live stock markets with real market data according to the principle of time-driven simulation. FinRL library strives to provide trading environments constructed by six datasets across five stock exchanges.

STATE SPACE, ACTION SPACE, AND REWARD FUNCTION

We give definitions of the state space, action space and reward function.

State space S. The state space describes the observations that the agent receives from the environment. Just as a human trader needs to analyze various information before executing a trade, so our trading agent observes many different features to learn better in an interactive environment. We provide various features for users:

• **Balance $b_t \in$ R+:** the amount of money left in the account at the current time step t.
• **Shares own $h_t \in$ Z n +:** current shares for each stock, n represents the number of stocks.
• **Closing price $p_t \in$ R n +:** one of the most commonly used features.
• **Opening/high/low prices o_t, h_t,$l_t \in$ R n +:** used to track stock price changes.
• **Trading volume $v_t \in$ R n +:** total quantity of shares traded during a trading slot.
• **Technical indicators:** Moving Average Convergence Divergence (MACD) Mt ∈ R n and Relative Strength Index (RSI) Rt ∈ R n +, etc.
• **Multiple-level of granularity:** we allow data frequency of the above features to be daily, hourly or on a minute basis.

Action space A. The action space describes the allowed actions that the agent interacts with the environment. Normally, $a \in A$ includes three actions: $a \in \{-1, 0, 1\}$, where $-1, 0, 1$ represent selling, holding, and buying one stock. Also, an action can be carried upon multiple shares. We use an action space $\{-k, ..., -1, 0, 1, ..., k\}$, where k denotes the number of shares. For example, "Buy 10 shares of AAPL" or "Sell 10 shares of AAPL" are 10 or -10, respectively.

Reward function $r(s, a, s_0)$ is the incentive mechanism for an agent to learn a better action. There are many forms of reward functions. We provide commonly used ones as follows:

• The change of the portfolio value when action a is taken at state s and arriving at new state s_0, i.e., $r(s, a, s_0) = v_0 - v$, where v_0 and v represent the portfolio values at state s_0 and s, respectively.

• The portfolio log returns when action a is taken at state s and arrives at new state s_0 i.e., $r(s, a, s_0) = \log(v_0 v)$.

• The Sharpe ratio for periods $t = \{1, ..., T\}$, i.e., $S_T = \text{mean}(R_t) \text{ std}(R_t)$, where $R_t = v_t - v_t - 1$.

• FinRL also supports user defined reward functions to include risk factor or transaction cost term.

DEEP REINFORCEMENT LEARNING AGENTS

The FinRL library includes fine-tuned standard DRL algorithms, namely, DQN , DDPG , Multi-Agent DDPG , PPO, SAC, A2C and TD3. We also allow users to design their own DRL algorithms by adapting these DRL algorithms, e.g. Adaptive DDPG, or employing ensemble methods. The implementation of the DRL algorithms are based on OpenAI Baselines and Stable Baselines.

BACKTESTING WITH TRADING CONSTRAINTS

In order to better simulate practical trading, we incorporate trading constraints, risk-aversion and automated backtesting tools.

Automated Backtesting plays a key role in performance evaluation. Automated backtesting tools are preferable because it reduces human error. In the FinRL library, we use the Quantopian pyfolio package [39] to backtest trading strategies. This package is easy to use and consists of various individual plots that provide a comprehensive image of the performance of a trading strategy.

Incorporating Trading Constraints. Transaction costs incur when executing a trade. There are many types of transaction costs, such as broker commissions and SEC fee. We allow users to treat transaction costs as a parameter in our environments:

• **Flat fee:** a fixed dollar amount per trade regardless of how many shares traded.
• **Per share percentage:** a per share rate for every share traded, for example, 1/1000 or 2/1000 are the most commonly used transaction cost rate for each trade.

Moreover, we need to consider market liquidity for stock trading, such as bid-ask spread. Bid-ask spread is the difference between the prices quoted for an immediate selling action and an immediate buying action for stocks. In our environments, we can add the bid-ask spread as a parameter to the stock closing price to simulate real world trading experience.

IMPLEMENTATION

[PLEASE REFER NOTEBOOK FOR COMPLETE IMPLEMENTATION DETAILS]

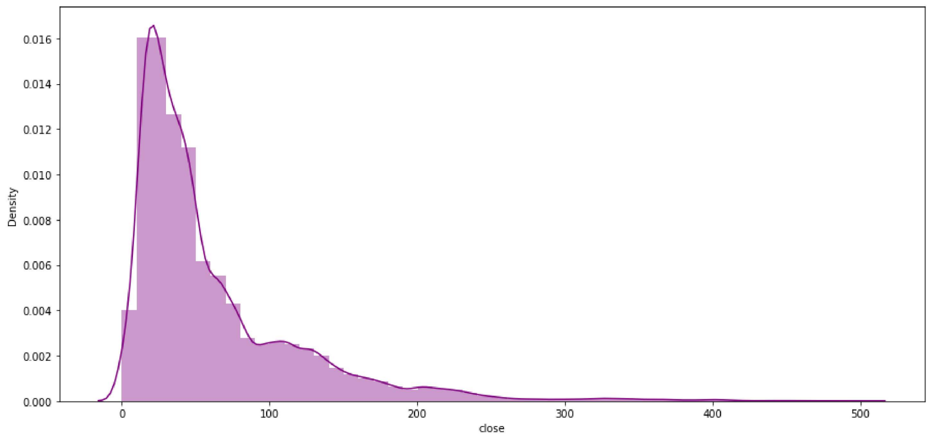
Assume we are at time t, at the end of day at time t, we will know the open-high-low-close price of the Dow 30 constituents stocks. We can use this information to calculate technical indicators In Reinforcement Learning we call these data or features as “states”.

1. We know that our portfolio value V(t) = balance (t) + dollar amount of the stocks (t).
2. We feed the states into our well trained DRL Trader, the trader will output a list of actions, the action for each stock is a value within [-1, 1], we can treat this value as the trading signal, 1 means a strong buy signal, -1 means a strong sell signal.
3. We calculate k = actions *h_max, h_max is a predefined parameter that is set as the maximum number of shares to trade. So we will have a list of shares to trade.
4. The dollar amount of shares = shares to trade* close price (t).
5. Update balance and shares. These dollar amounts of shares are the money we need to trade at time t. The updated balance = balance (t) −amount of money we pay to buy shares +amount of money we receive to sell shares. The updated shares = shares held (t) −shares to sell +shares to buy.
6. So we take actions to trade based on the advice of our DRL Trader at the end of day at time t (time t’s close price equals time t+1’s open price). We hope that we will benefit from these actions by the end of day at time t+1.
7. Take a step to time t+1, at the end of day, we will know the close price at t+1, the dollar amount of the stocks (t+1)= sum(updated shares * close price (t+1)). The portfolio value V(t+1)=balance (t+1) + dollar amount of the stocks (t+1).
8. So the step reward by taking the actions from DRL Trader at time t to t+1 is r = v(t+1) − v(t). The reward can be positive or negative in the training stage. But of course, we need a positive reward in trading to say that our DRL Trader is effective.

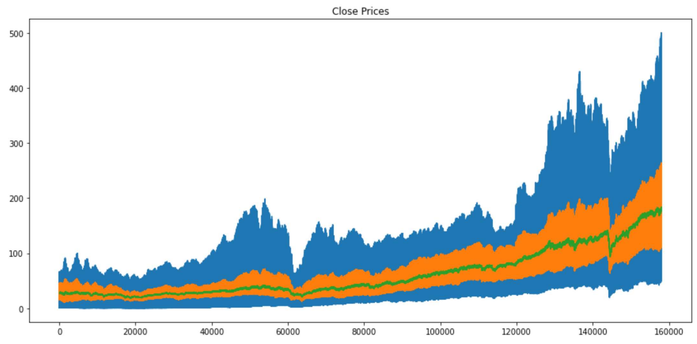
9. Repeat this process until termination.

RESULTS

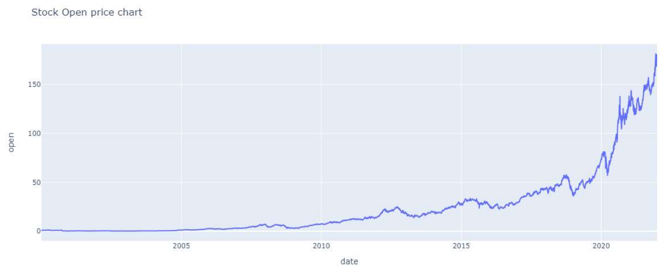
1) **Analysis on Trends in Stocks Data:** We tried to analyze trends in open and close price of 5 stocks:

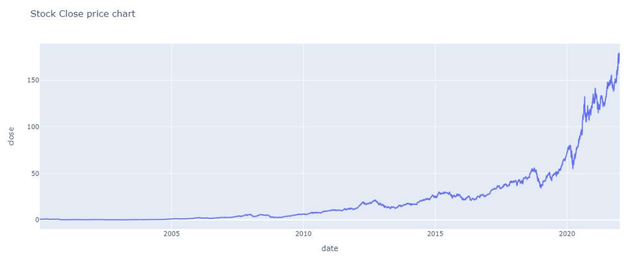


2) Close Prices on all stocks:

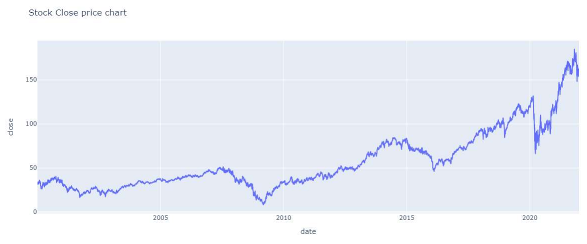


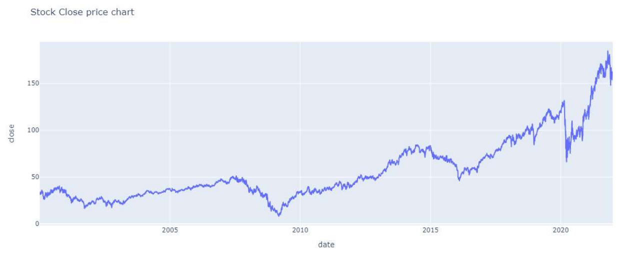
3) Open and Close Price chart of Apple:



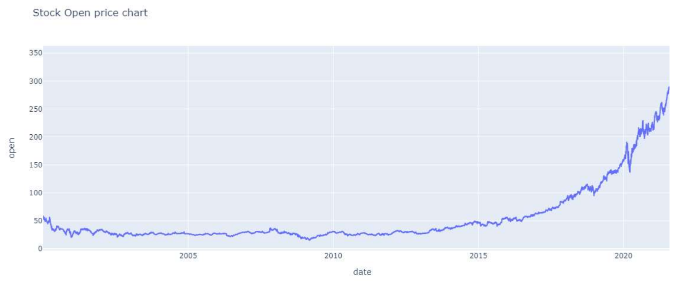


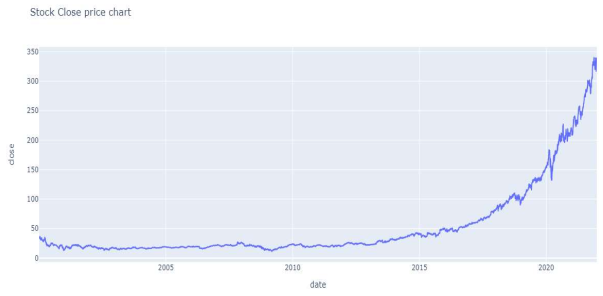
4) **Open and Close Price chart for AXP:**





5) Open and Close Price chart for MSFT:





RESULTS USING DDPG:

We use pyfolio to obtain following information:

There were a couple of worst phases in the US market, where the market performed badly (Markets were down 20-50%) and we wanted to capture those trends so we took 20 years of data in order for our model to learn and predict the worst market of the future if any. This might have affected the sharpe ratio and training of our model but we were able to capture the trends and out model returned more than 5% average returns in 2020 when the covid hit and markets were slumped.

Start date	2000-01-03
End date	2018-12-30
Total months	330
	Backtest
Annual return	5.212%
Cumulative returns	304.973%
Annual volatility	14.795%
Sharpe ratio	0.42
Calmar ratio	0.12
Stability	0.87
Max drawdown	-42.695%
Omega ratio	1.10
Sortino ratio	0.60
Skew	NaN
Kurtosis	NaN
Tail ratio	0.97
Daily value at risk	-1.839%

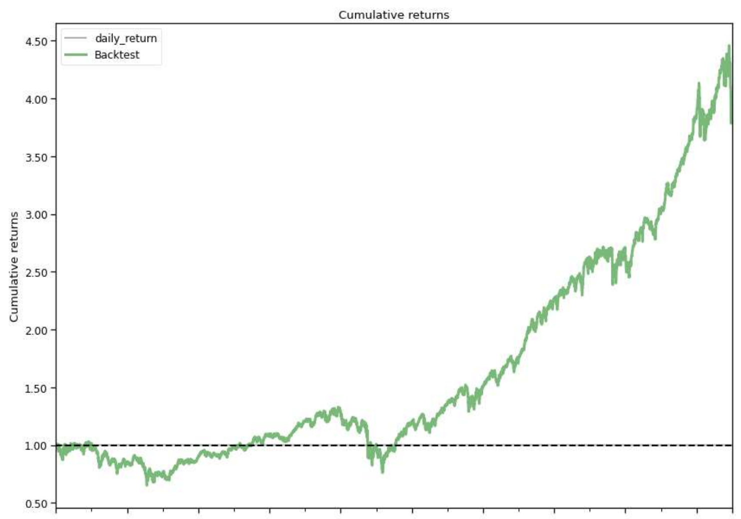
```
Mean of empty slice
```

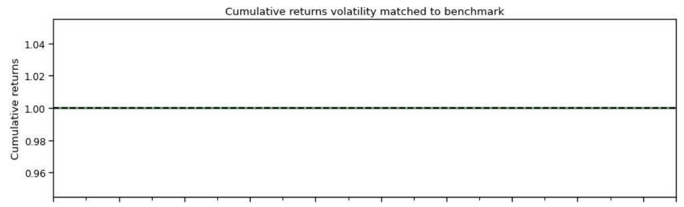
Stress Events	mean	min	max
Dotcom	0.06%	-5.36%	5.99%
Lehman	-0.03%	-8.51%	5.72%
9/11	-0.01%	-4.28%	1.98%
US downgrade/European Debt Crisis	0.04%	-5.83%	4.22%
Fukushima	0.10%	-1.78%	1.36%
US Housing	-0.22%	-2.81%	2.11%
EZB IR Event	0.06%	-0.75%	1.69%
Aug07	0.07%	-2.34%	2.27%
Mar08	0.05%	-2.23%	3.28%
Sept08	-0.13%	-8.51%	5.72%
2009Q1	-0.36%	-4.39%	2.82%
2009Q2	0.27%	-3.94%	6.59%
Flash Crash	-0.21%	-3.24%	3.45%
Apr14	-0.03%	-2.35%	1.32%
Oct14	0.13%	-2.13%	2.07%
Fall2015	-0.18%	-4.20%	4.41%
Low Volatility Bull Market	0.02%	-3.11%	2.68%
GFC Crash	-0.04%	-8.51%	11.57%
Recovery	0.05%	-5.83%	4.22%
New Normal	0.04%	-4.62%	4.46%

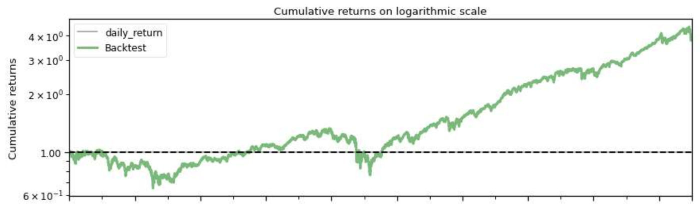
Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	42.69	2007-12-10	2009-03-07	2010-12-02	779
1	36.69	2000-11-29	2002-07-23	2005-07-15	1208
2	15.21	2011-07-07	2011-08-10	2012-01-10	134
3	15.14	2018-12-03	2018-12-22	NaT	NaN
4	13.47	2000-01-14	2000-03-14	2000-06-02	101

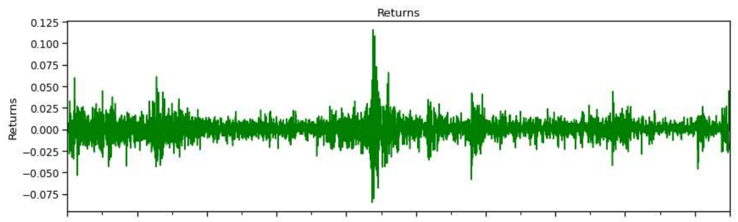
1) **Returns Plots**

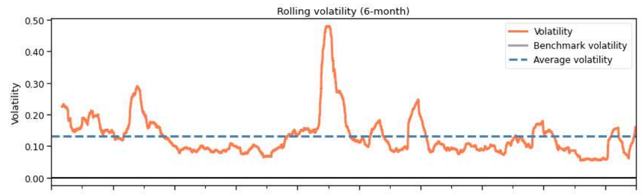
Plots of cumulative returns and daily, non-cumulative returns allow you to gain a quick overview of the algorithm's performance and pick out any anomalies across the time period of the backtest. The cumulative return plot also allows you to make a comparison against benchmark returns - this could be against another investment strategy or an index like the S&P 500.

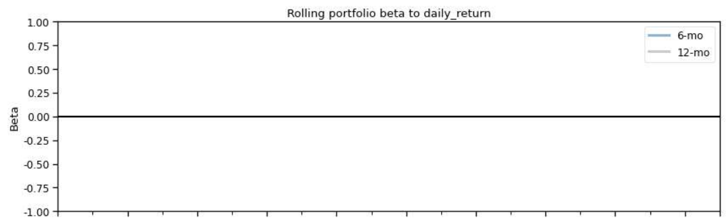


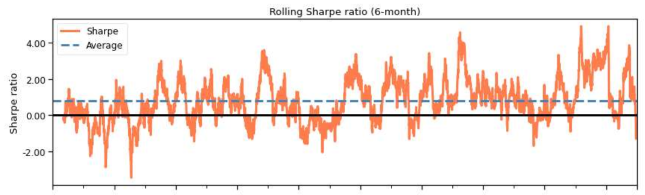




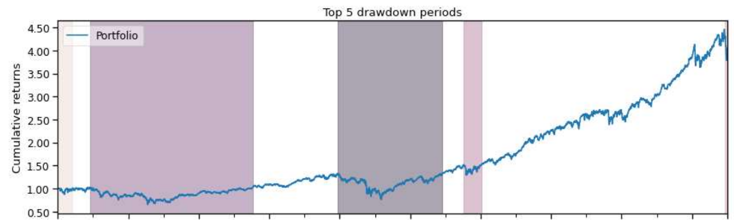


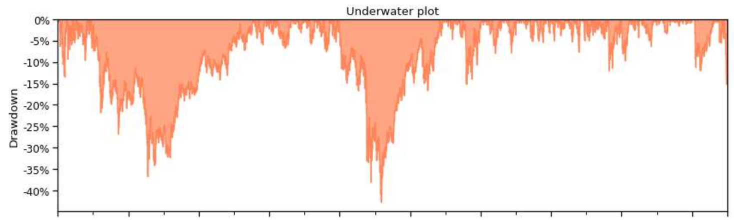


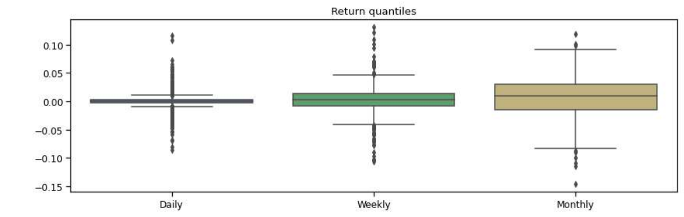


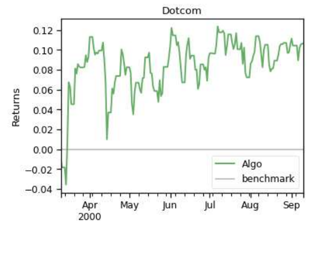


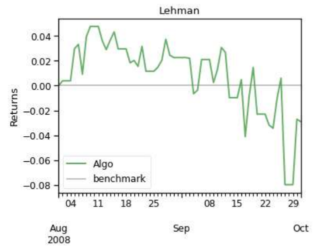
With the annual and monthly return plots, you can see which years and months the algorithm performed the best in. In a backtest with a longer period of time, these plots will reveal more information. Furthermore, the distribution of the monthly returns is also instructive in gauging how the algorithm performs in different periods throughout the year and if it is affected by seasonal patterns.











Return Quantiles

These box and whisker plots provide an overview of the return quantiles broken down by the return timeframe (daily / weekly / monthly) across the entire backtest time period.

The center line in the middle of each box shows the median return, and the box shows the first quartile (25th percentile) as well as the 3rd quartile (75th percentile). While a high median return is always helpful, it is also important to understand the returns distribution. A tight box means that the bulk of the returns (25th - 75th percentile) fall within a tight bound - i.e. the returns are consistent and not volatile. A larger box means that the returns are more spread out. It is important, however, to take note of the scale to the left to put the quartiles in perspective. In addition, returns over longer periods of time will have a wider distribution as increasing the length of time increases the variability in returns.

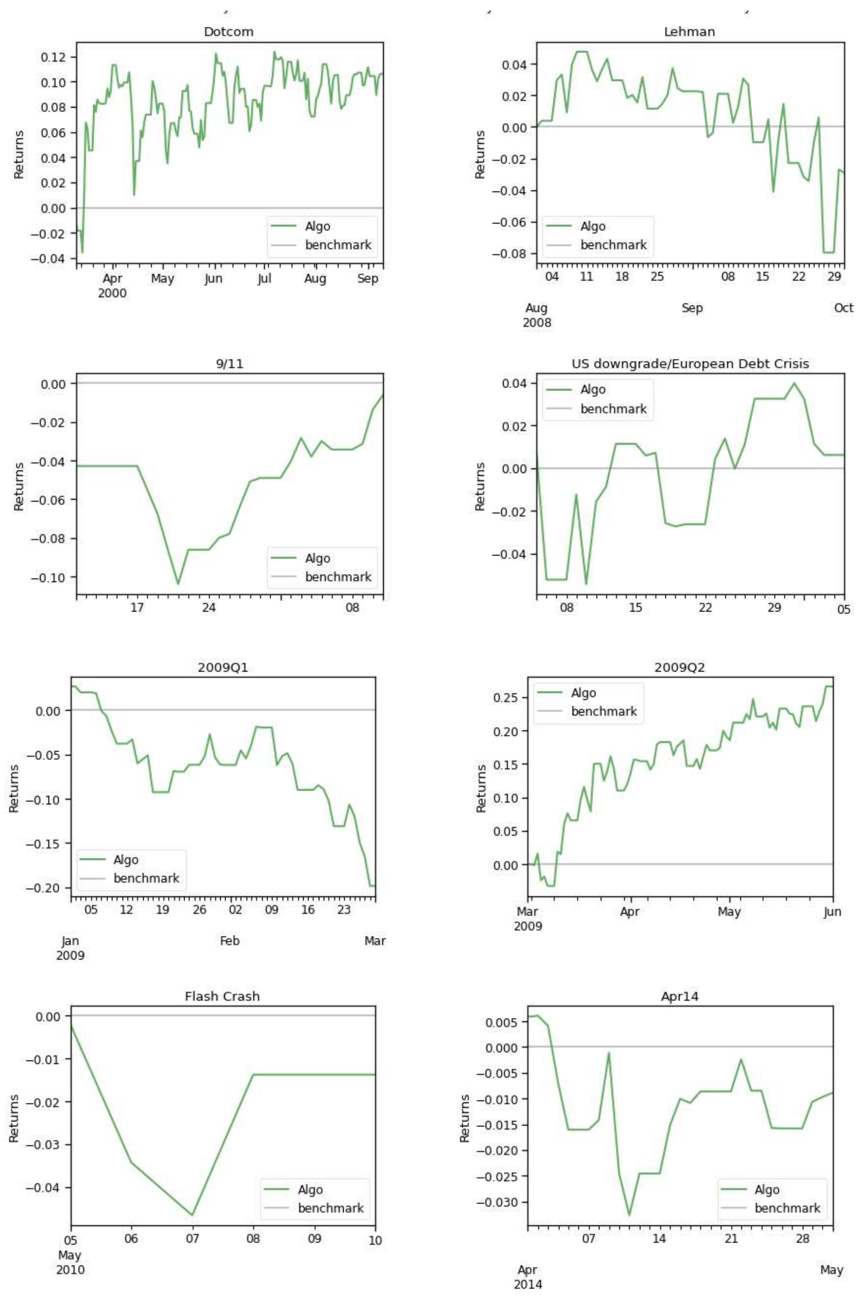
The 'whiskers' at the end indicate the returns which fall outside the 25th and 75th percentile. A tight box with long whiskers indicate that there may be outliers in the returns - which may not be ideal if the outliers are negative. This may indicate that your strategy may be susceptible to certain market conditions / time periods.

Rolling Plots

Below, we have several rolling plots which show how an estimate changes throughout backtest period. In the case of the rolling beta and the rolling Sharpe ratio, the rolling estimate gives us more information than single point estimate for the entire period. A rolling estimate allows the user to see if the risk-adjusted return of the algorithm (Sharpe ratio) is consistent over time or if it fluctuates significantly. A volatile Sharpe ratio may indicate that the strategy may be riskier at certain time points or that it does not perform as well at these time points. Likewise, a volatile rolling beta indicates that it is exposed to the market during certain time points - if the strategy is meant to be market neutral, this could be a red flag.

Rolling Beta Plot

The plot below shows the rolling beta of the strategy against benchmark returns over the entire period of the backtest. In this instance, the benchmark return of the SPY was used. Thus, the lower the rolling portfolio beta to the SPY, the more market neutral an algorithm is.



Related Works

We have reviewed related works on relevant open source libraries and existing applications of DRL in finance.

Recent works can be categorized into three approaches: value based algorithm, policy based algorithm, and actor-critic based algorithm.

FinRL has consolidated and elaborated upon those algorithms to build financial DRL models. There are a number of machine learning libraries that share similar features as our FinRL library.

• OpenAI Gym is a popular open source library that provides a standardized set of task environments. OpenAI Baselines implements high quality deep reinforcement learning algorithms using gym environments. Stable Baselines is a fork of OpenAI Baselines with code cleanup and user-friendly examples.

• Google Dopamine is a research framework for fast prototyping of reinforcement learning algorithms. It features plugability and reusability.

• RLlib provides high scalability with reinforcement learning algorithms. It has modular framework and is very well maintained.

• Horizon is a DL-focused framework dominated by PyTorch, whose main use case is to train RL models in the batch setting.

REFERENCE

FinRL Architecture Description -
https://finrl.readthedocs.io/en/latest/start/three_layer/environments.html

https://openreview.net/forum?id=82nKnFErXUh}

https:www.wikipedia.com

Matthew F. Dixon Igor Halperin Paul Bilokon -
Machine Learning in Finance: From Theory to Practice