Master Thesis

Universitat Politècnica de Catalunya

Universitat de Barcelona

Universitat Rovira i Virgili

Reinforcement Learning for Portfolio Optimization

Rafael Bianchi

Supervisor: Mario Martin

April 20, 2022

Abstract

In this study, the potential of using Reinforcement Learning for Portfolio Optimization is investigated, considering the constraints set by the stock market, such as liquidity, slippage, and transaction costs. Five Deep Reinforcement Learning (DRL) agents are trained in two different environments to test the agents' ability to learn the best trading strategies to allocate assets, expecting to generate higher cumulative returns. All agents used are model-free and already optimized for financial problems, using the FinRL library. Therefore, the state-space has a high dimension, as found in the financial market environments. The two proposed environments use market data from US stocks, and one of them also uses Finsent data, an alternative data source that contains the news sentiment for all the stocks that are part of Dow Jones Industrial Average (DJIA). A series of backtesting experiments were performed from the beginning of 2019 to the beginning of 2020 and compared the two environments and how the agents performed against the DJIA. All the results were assessed with the pyfolio Python library, which uses all standard metrics to evaluate portfolio performance. Some algorithms increased the cumulative returns compared to the first dataset. The best result obtained outperformed DJIA by a significant amount and a smaller drawdown.

Keywords: Reinforcement Learning; Machine Learning; Portfolio Optimization; Finance; Machine Learning; Markov Decision Processes

Acknowledgement

I would like to thank Professor Mario Martin for guiding me through this project. His support was fundamental to challenging me and helping me expand my knowledge on the subject. Beyond the tutoring, Professor Mario Martin impacted my trajectory in this Master's substantially by being an excellent reference for a future I dream of and empathic to the challenges in my personal life.

I also would like to thank my family for their continuous support in this journey. I would like to thank my wife for encouraging me to pursue a career change, my parents for investing in their children's education, my sister for visiting me in Barcelona when I most needed it, my sister-in-law for the guidance in the academia, and my work colleagues for the lifetime learning on technology.

Lastly, I would like to thank the University for the opportunity to learn and be part of a great group of minds thinking not only about this exciting subject of artificial intelligence but also about how early technologies can play an essential role in contributing to shaping a more ethical and better world for all humans.

List of Figures

List of Tables

Listings

Acronyms

A2C Advantage Actor Critic. vi, 31, 35, 42

AAPL Apple Inc.. vii, 27

AI Artificial Intelligence. 1, 6

B3 Brasil, Bolsa, Balcão. 23

BPN Back-propagation Neural Network. 22

CLA Critical Line Algorithm. 21

DDPG Deep Deterministic Policy Gradient. vi, 23, 31, 35, 36, 38

DJIA Dow Jones Industrial Average. iii, 2, 32, 46

DL Deep Learning. 22, 23

DRL Deep Reinforcement Learning. iii, 23, 35

EOD End of Day. 45

IPO Initial Public Offering. 46

MDP Markov Decision Processes. 3, 6, 8–11, 27, 28

ML Machine Learning. 1, 22, 23, 44

MP Markov Process. 6, 8, 9

MPT Modern Portfolio Theory. 15

MRP Markov Reward Process. 8, 9

NNF Neural Network with Fixed noise. 22

NYSE New York Stock Exchange. 1

OHLCV Open, High, Low, Close and Volume. 23, 27, 35

PPO Proximal Policy Optimization. vi, 31, 35, 39, 40

RL Reinforcement Learning. 1–3, 6, 10–13, 23, 25, 27, 28, 45, 46

SAC Soft Actor-Critic. vi, 31, 35, 41

SVM Support-vector Machine. 22

SVR Support-vector Regression. 22

TD3 Twin Delayed DDPG. vi, 31, 35–37

Contents

Chapter 1

Introduction

The stock market and computer technology have worked together for a long time. For example, the New York Stock Exchange (NYSE) has been computerized going back to 1966, even though the first trading, made by an automated system, only occurred in 1982.

Nowadays, computers handle all the transactions, and the financial industry constantly tries to keep up with new technologies. In this context, computers technology has been a differential for the top players by improving their returns and helping to mitigate their losses.

Several different methods and systems have been used in financial market applications, including control algorithms, signal processing, and statistical/mathematical methods. Given the success and growing popularity of Machine Learning (ML), the use and the traction of ML models also got the interest of the financial markets.

In this thesis, we aim to explore portfolio optimization using Reinforcement Learning (RL), a branch of Artificial Intelligence (AI) that solves the sequential decision-

making process, directly interacting with the environment.

In this introductory chapter, we define the goals of the thesis and highlight the definition of the problem and the motivations.

1.1 Problem Definition

The aim of the portfolio optimization problem is to find the best asset allocations (weights) that maximize long-term return.

This can be mapped to an RL terminology, using historical stock prices, news sentiment, and the current portfolio (e.g., weights for each asset, technical indicators, prices, etc.) as states, modeling it to a model-free RL so that the agent finds the optimal policy and maximizes the returns.

The problem in this report is set as trying to follow or beat the DJIA, composed of thirty stocks. A successful RL solution is defined as when an agent can manage this portfolio, performing the actions of buying, selling, or holding the stocks and maximizing the rewards in the long term.

All possible tradeable assets actions must be taken only once per day, as day trading is not allowed. The actions are discrete and have a pre-defined boundary limiting how many shares can be bought or sold.

1.2 Motivation

An attempt to experiment RL framework in portfolio management problems is motivated by the successful implementations and advances made in the RL field. These implementations range from various domains such as video games (Mnih et al., 2015), board gaming (Silver et al., 2016), and protein folding (Callaway,

2020).

However, the number of research about RL in the financial domain is still low (growing a lot lately, though).

1.3 Structure of the Report

The report is organized into three parts: Background (Part I), Literature Review (Part II), and Experiments (Part III). A brief outline of the project structure and chapters is provided as follows:

Chapter 2: Reinforcement Learning This chapter presents a brief introduction for RL main components and the Markov Decision Processes (MDP) framework.

Chapter 3: Financial Foundations This chapter introduces the basic financial terms and concepts of the method developed later in the report.

Chapter 4: Portfolio Optimization In this chapter, we discuss some mathematical, systematic, and automated methods for optimizing a portfolio. Related work for each of the method types is also presented here.

Chapter 5: Materials and Methods Materials and methods that will be used in the experiments are presented in this section, with an explanation of the FinRL framework and the assumptions that will be considered for the training and trading part. Also, the format and previews of the datasets that are used in this report.

Chapter 6: Results This chapter presents the results and evaluates the performance of the trading agents in two scenarios: simple market data and indicators and market data with Finsent data.

Chapter 7: Conclusion The conclusion of this report, considering the limitations and future work that could improve the performance of the trading agents.

Part I

Background

Chapter 2

Reinforcement Learning

RL is a subtopic of AI based on MDP, and it is also known as approximated dynamic programming. The goal is that the agent learns, by interacting with the environment, which action to take in a given observed state and maximize the expected cumulative reward. Based on a simple action-feedback cycle, the agent is able to actively adapt to the environment.

This chapter addresses some mathematical representations, formalisms, the concepts of MDPs, and it also reviews the main components of RL.

2.1 Markov Processes

Markov Process (MP) is a stochastic process that is also known as the Markov chain. The system is formalized with states, which are observations that can be observed from the state space and a time index (discrete or continuous). The state space is set as all possible states from the system. A sequence of states (or chain) over time is referred to as history. In order to be defined as a Markov process, the Markov property (equation 2.1) must be satisfied. The Markov property requires

that a state s at time t (S_t) must contain all information to model the system's future dynamics and do not depend on past states. In other words, each state s must be unique and distinguishable from each other. A system that complies with the Markov property can utilize the transition probabilities from a transition matrix, a square matrix with $N \times N$, where $N = size$ (state-space).

$$P_{ss}^{a} = Pr\{s_{t+1} = s^{t} | s_t = s, a_t = a\} \; \forall s, s^{t} \in S, a \in A(s) \tag{2.1}$$

As Figure 1 illustrates, the MP can be represented as a graph with nodes corresponding to the states and the directed edges representing the transition probability from a state to another.

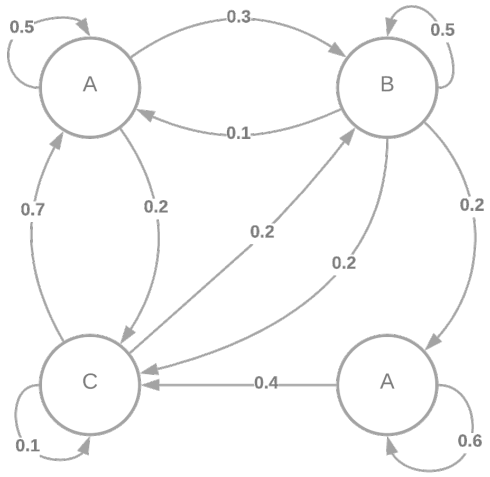

Figure 1: MP graph representing the state transition with probabilities.

The Markov property also implies that the environment is stationary, meaning that the states' transition distribution does not change over time, regardless of the transition history.

2.2 Markov Reward Processes

The Markov Reward Process (MRP) extends MP and adds the reward to the transition between states. The reward is usually defined as a square matrix, similar to the transition matrix, with a reward given for a transition from S_t to S_{t+1}. Rewards might be positive or negative (penalizing) and usually are defined as natural numbers. The total reward, for a given time t, is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

As noted in equation 2.2, a new factor gamma is added that defines the importance that the agent gives to future rewards. With that, future rewards distant from t are multiplied by the discount factor gamma raised by the time steps distant from t. The value of gamma ranges from 0 to 1. When gamma = 0, the agent is short-sighted and does not take the future rewards into account, and, when gamma = 1, all future rewards are considered to compose the return at time t (G_t). The gamma is also helpful because the sum of the future states might be infinite, as there might be transitions that can be visited a countless number of times. In practice, the G_t value is hardly used as it can vary widely, even for the same state. Instead, the value of the state $V(s)$ (equation 2.3) is used. The value for every state is the expected (average) return following the MDP.

$$V(s) = E[G|S_t = s] \tag{2.3}$$

The MP is modified, and the reward value is added to the transitions, as shown in Figure 2.

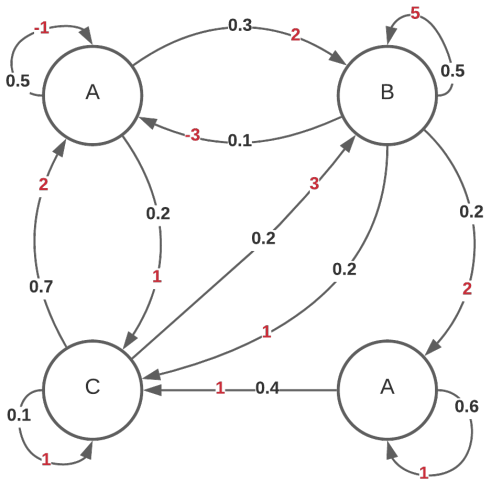

Figure 2: MRP graph with probability transitions (black) and rewards (red).

2.3 Markov Decision Processes

As noticed in the previous models (i.e., MP and MRP), they can only be used to observe the environment passively, getting some probabilities to infer future states. In order to define the MDP, the MRP needs to be extended in a way that the agent can interact with the environment. This extension is a set of actions (A), and the transition matrix now has to consider the action taken in one state getting to another. When the agent chooses an action, the agent affects the transiting probabilities to a target state. Now, the agent's rewards depend on the state and the action taken in a state S_t.

This modification adds an extra dimension to the transition matrix, changing the dimensions to $NxNxA$ with N as the number of states and A as the number of actions.

A finite discrete-time fully observable MDP is a tuple $\langle S, A, D, T, R \rangle$, where:

- S - it is a finite set of possible states in an environment.
- A - finite set of actions that an agent can take
- D - Sequence of natural numbers $(1, 2, \ldots, T_{max})$, discrete epochs in which actions are taken.
- T - The transition function gives the probability of ending up in a state S_2, given an action taken in a state S_1.
- R - It is the reward function that returns a real number, positive when it is modeled as a reward or negative when it is modeled as cost.

2.4 Components of Reinforcement Learning

RL is a framework used to solve problems that are described in an MDP model. The RL relies on the problem being well fitted and modeled correctly to the MDP model, even though all the transitions do not need to be known. On the other hand, a problem that cannot be described as an MDP might not be suitable for a RL framework. Following the definitions of the MDP, the RL usually have four main components: rewards, value functions, policy, and environment.

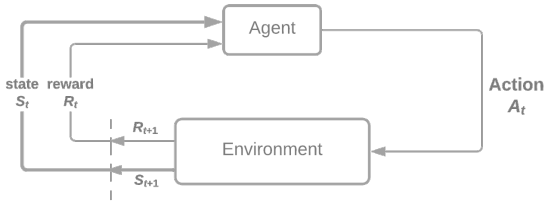

Figure 3: The agent-environment reward process.

2.4.1 Policy

With a full formalized MDP, the last thing and one of the most important items to cover of an RL or MDP problem is the policy. The policy is the map between states and actions that control the behavior of an agent in a Markovian Process. It can be considered the agent's knowledge through interaction with the environment by trial and error, or by acting and being rewarded. Mathematically, the policy is the likelihood of actions for every possible state (equation 2.4), that is the probability of taking action a_t, given a state s_t.

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2.4}$$

2.4.2 Rewards

The reward function returns a scalar value which hints to the agent how good or bad it is behaving. It is the goal of RL to maximize the total reward received by the agent over a period of time. The reward is the main reason for changing a policy as if an action executed by following the policy gives a bad or low reward, the next time that the agent faces the same states, the policy might return another action to be executed.

2.4.3 Value function

If the reward function gives the value of the action that has been just executed, the value function valuates in the long run. The value function determines how good is the state and it is a more fair-sighted approach to learn how good is the environment in a state s_t.

2.4.4 Environment

The last component of the RL system is the representation of the environment. This representation is close as possible to the behavior and dynamics of the environment that allows the agent to infer the next state, given a state and an action. RL methods that use models are called model-based and the ones that use the trial and error approach are called model-free.

Chapter 3

Financial Foundations

Portfolio optimization is the method of picking the best asset distribution in a portfolio to maximize the expected return. This optimization usually considers the assets' value and other factors as transaction costs, taxes, and other constraints. Investment managers try to put effort using their knowledge, forecasts and insights in building a portfolio. This chapter aims to address the data structure of financial data, Financial Terms & Concepts, and the link between the trading signals in finance and RL's possible applications that can be used to create dynamical systems.

3.1 Financial Terms & Concepts

To better connect the ideas from finance to RL, this section has the definitions for the financial terms and concepts.

3.1.1 Asset

An asset is any object (tangible or intangible) with a value to its holder (Jonathan Law, 2008). It can be cash itself or loans, stocks, rates, and, in most circumstances, can be converted to cash. Of course, there are many other types of assets like patents, trademarks, buildings, but our focus in this report is basically on stocks and cash.

3.1.2 Portfolio

A portfolio is the set of holdings in securities owned by an investor or institution (Jonathan Law, 2008). This set of holds are financial assets, and it is defined as the vector:

$$w_t = [w_{1,t}, w_{2,t}, \ldots, w_{M,t}]^T \in \mathbb{R}^M \tag{3.1}$$

In equation 3.1, M is the number of assets the portfolio has, the w_t represents the portfolio at the time t, and each i-th represents the ratio of the total invested. These ratios have to sum to 1, as equation 3.2 shows.

$$\sum_{i=1}^{M} w_{i,t} = 1 \tag{3.2}$$

Investments portfolios are a powerful tool to reduce risk exposure as the number of assets, and sector distribution, are related to lower exposure risks. For example, portfolios of 10 stocks (also distributed in various sectors) are less risky than portfolios with two stocks (Chen, 1999).

3.1.3 Portfolio Optimization

Portfolio optimization is a process to obtain the weights for each asset under the portfolio that maximizes the portfolio value. In 1952, Harry Markowitz created a mathematical framework called Modern Portfolio Theory (MPT). This mathematical framework works by building portfolios to maximize the expected return for a given level of risk. Practically, a single asset risk should not be analyzed alone, and it should be analyzed how it performs relative to other assets in the portfolio. Ideally, the positive performance of one asset should cancel the negative performance of others.

3.1.4 Short Selling

Short selling is a strategy used when the portfolio manager believes in declining the asset's price. It is also used for hedging strategy for a long position (for the same asset) to reduce exposure. The process of short selling is a little bit more complicated than a long position because the investor has to borrow assets, sell them, and repurchase them at a lower price before giving them back to the original owner, thus, making some profit.

3.2 Financial Data Structures

Financial data is any information(price, analysts' opinions, signals, graphs, and others) used to analyze and create trading strategies in a trading platform. Most of the financial data available are unstructured and cannot be used directly by ML without preprocessing. As with any data input for ML applications, the data's quality is essential for the precision of the ML application's prediction and the policies that will be improved by the RL application.

3.2.1 Data Types

There are essentially four types of financial data (see table 1): fundamental data, market data, analytics, and alternative data. They have different degrees of diversity and complexity. The value of the public data (e.g., stock prices) has the same value for everyone using it, as everyone has the same data. The value of private datasets (e.g., alternate data) is quite the opposite, which offers a unique opportunity, as the other competitors do not have access to it.

Table 1: The four essential types of financial data

Fundamental Data	**Market Data**	**Analytics**	**Alternative Data**
Assets	Price/yield/implied volatility	Analyst recommendations	Satellite/CCTV images
Liabilities	Volume	Credit ratings	Google searches
Sales	Dividends/coupons	Earnings expectations	Twitter/chats
Costs/earnings	Open interest	News sentiments	Metadata
Macro variables	Quotes/cancellations	...	...
...	Aggressor side		
	...		

Note. Reprinted from de Prado (2018, p. 24)

Fundamental Data

Fundamental data is the data that is released when the company has to comply with regulatory guidelines. This data is usually public and is composed of accounting data reported quarterly. When using fundamental data, the main problem is that the data released publicly is not the same as when it was created. This time difference might compromise the prediction, results and allocation planning for the portfolio. Another problem with the fundamental data is that the reports' value might get fixed and another version released later, and the RL (or ML) application has to be ready to use either version at the proper time. Even though the fundamental data has a low frequency and is available to everyone in the market,

it is hard to get an advantage as there is no much left unexploited. It might be used with other data types and joined to get insightful information for choosing the portfolio's assets.

Market Data

Market data is all the trade-related data for an asset that is reported by an exchange. This sort of data is very time-sensitive and abundant, with terabytes generated daily. Every market participant leaves a footprint in the trading records, making it possible for competitors to rebuild the trad anticipate and market. The structure of the raw data feed is unstructured (FI/X messages) and has to be preprocessed. This preprocessing is not complicated, but it might require high hardware resources to keep up with the market as it is time-sensitive. The historical market data is also essential to forecast prices and estimate uncertainty on portfolios.

Analytics

According to the source data, analytics is the derivative data based on alternative, fundamental, or market data. It can also be a combination of many data sources. Original data has to be processed by something or someone else and then made available to be considered analytics data. Banks usually generate this type of data, analyzing the source data and creating an in-depth analysis of companies, countries, and governments and then selling to consumers. There are some pros and cons when buying alternative data. Even though the data is not openly available, which increases its uses and intrinsic value for the consumer, the data might be biased, and the criteria for transforming it might not be known.

Alternative Data

Alternative data is the information about an organization created or made available from sources outside the organization. The type of data (image, text, news, and others) is broad, usually unstructured. This information creates timely opportunities for investments, and it is very promising for use in ML systems as ML systems might find useful patterns for better predictions. Combining other financial data types creates opportunities for portfolio managers to create better portfolio optimization strategies that can increase the expected return in the long run.

3.3 BARS

As for most Machine Learning solutions, the input must be regularized, usually available in a table format. This kind of format is known in finance as BARS. There are two types of bars: standard bars and information-driven bars. Forming bars and understanding their use and common pitfalls is an important part of the process of trading.

3.3.1 Standard Bars

This type of bar construction is highly available in the financial industry, and they are frequently used by algorithmic trading applications and for trader analysis. Several vendors make it usable through APIs, which makes it easier to be used across the globe.

Time Bars

The most common bars are formed by sampling data on a fixed time frame interval (e.g., hourly or daily) and habitually carry information about the open/close and

low/high prices in the chosen interval. This kind of bar should be used with care as it might downsample data around the market's most active hours (opening/closing periods) and oversample data around the market's most inactive hours.

Tick Bars

Tick bars are a straightforward method that groups the variables (open, close, high, low volume, and others) after a pre-defined number of transactions. This type of bar makes it easy to synchronize data arrival but comes with the downside that it might introduce outlier as it could compress large orders in one tick and a large number of small order in another tick.

Volume Bars

As the name implies, volume bars sample the information after a certain volume of the security's units have been exchanged. This method circumvent the problem of order fragmentation that tick bars might introduce to the algorithm.

Part II

Literature Review

Chapter 4

Portfolio Optimization

Portfolio construction is one of the most known financial problems. As a result, investment managers, economists, and academics have been spending decades trying to find the best tool and methods to build a portfolio computationally efficient. As introduced in subsection 3.1.3, Portfolio Optimization is the process of finding the best portfolio (or best asset distribution) according to an objective function. This chapter explores the current methods and algorithms that address the portfolio management problem.

4.1 Quadratic Programming

A few quadratic optimization methods are specifically developed for portfolio optimization. One of these methods is famous because Markowitz himself designs it. Markowitz developed the Critical Line Algorithm (CLA) (open-source implementation by Bailey and López de Prado (2013)), a quadratic optimization procedure that is guaranteed to achieve the exact solution after a determined number of iterations. Besides the specificity of the algorithm, and the guarantee to find the

solution, it has some caveats that might make it unreliable. For instance, if the predicted returns have slightly high deviations, it might create different portfolios (Michaud, 1998).

4.2 Machine Learning

ML models are based on statistical and mathematical methods that have been used for decades. This assumption makes ML a good start for predicting anything, from churn rate to stock prices. Kim (2003) presented a study about using a Support-vector Machine (SVM) for predicting an asset price direction and how to find the C bound and kernel parameters as they are very critical for the results. This model was also compared with other models (Back-propagation Neural Network (BPN) and SVM), showing that the SVM can be used as an alternative to other ML methods for predicting stock prices. Meesad and Rasel (2013) also used an Support-vector Regression (SVR) and different windowing operators as the algorithm's input to predict stock trends and prices. These solutions are model-based and have another step to use the predictions for the trading part.

4.3 Deep Learning

Besides the enormous advances and increased use of Deep Learning (DL) to solve vast types of problems, there are few portfolio optimization problems solved using deep learning (Kwak et al., 2021). Yun et al. (2020) proposed a two-staged DL framework that addresses the relative and absolute returns using a cost function. This proposed method outperforms ordinary single-staged DL process in all tests. Kwak et al. (2021) also used a DL model to optimize the portfolio. Using a Neural Network with Fixed noise (NNF), the model outperformed the baselines, defined as an equally-weighted portfolio and Shallow NNF. Even though they demonstrated

DL models that can optimize portfolios, they are model-based methods and do not do the trading part directly. Rather, they require further methods to process the predicted prices, which generally use traditional heuristics. Both ML and DL methods do not consider the transaction costs, and it is an essential constraint for portfolios built on markets that charge these costs from the transactions.

4.4 Reinforcement Learning

The RL solutions have been getting traction in the financial industry lately. Due to the nature of the RL, which tries to find the best actions to maximize the rewards (or minimize costs), it seems fit for optimizing a portfolio.

Martinez and Pereira (2020) presented a daily trading system based on the DDPG agent to optimize the portfolio allocation problem. Their environment is based on the Brasil, Bolsa, Balcão (B3) stock exchange and has some different assumptions: (i) a set of ten stocks is bought and held until the end of the day; (ii) they always sell the stocks within the closing action; (iii) there is no overnight position.

They were able to generate alpha for three years, beating all the other B3 benchmarks (i.e., Ibovespa). The justification of their work with DRL agents instead of using DL or ML solutions was partly mentioned in 4.2 and 4.3, as these other solutions do not take some markets aspects into account (transaction costs, having another part for trading, etc.). They also added that the policies that the agent learns by interacting with the environment are so complex that it would be challenging for a human being to be able to write in a rule-based system.

Another DRL solution for the portfolio optimization problem was presented by da Silva (2021). This work was also based on the B3 exchange stock, but it also considered news sentiment, together with Open, High, Low, Close and Volume

(OHLCV) data. The author showed that the proposed system is ready to be implemented on commercial platforms for testing with all the market aspects and connecting it with brokerage services to let the agent make orders automatically and interact with the real environment.

Chapter 5

Materials and Methods

5.1 FinRL

In order to compare the state-of-the-art RL agents, we will use the FinRL(Liu et al., 2021) framework, an ecosystem framework designed to automate portfolio management tasks and includes various markets and state-of-the-art algorithms, which are also implemented in an optimized way for finance.

In this chapter, we explain and define the main components of the FinRL and how they are used. Firstly, we consider the necessary assumptions and requirements, followed by the framework that enables reinforcement learning agents to interact with the financial market to address portfolio management optimally.

5.1.1 Assumptions

No Slippage

Slippage is the difference between the expected price at which an order is placed and the actual price at which the trade occurs. It occurs when the order placed on the exchange is executed at a price different from the price requested.
Slippage usually happens in highly volatile markets, when asset prices change so often that it is difficult to put an order at a specific price and execute it, as the price might change quickly between the order being placed and executed. Slippage also happens when there is low liquidity in the market for an underlying asset due to the difficulty of finding an order matching the expected value for the buyers and sellers.
We assume that the liquidity of all our portfolio traded assets is high sufficient so the trades can be executed instantly with the last price.

Liquidity

Liquidity is used to define assets that can be converted to cash, with zero or slight loss. Following the previous assumption, we assume that there is enough liquidity to trade (buy or sell) an asset.

No Market Impact

When trading, it is expected that the bigger the volume being traded, it might affect the prices of the assets. We assume that the volume that our agents will trade is not significant to impact the market. Thus, our state-space will not include any information about market impact.

5.1.2 Market Environments

As with any RL solution, it requires that the data available is in a format that can be mapped to an MDP state space. The FinRL framework contains some market environments to pull the data(OHLCV) in a tabular form as shown in table 2.

Table 2: Sample data for 10 days for the AAPL ticker using the yfinance python library.

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2022-03-03	168.470001	168.910004	165.550003	166.229996	76678400	0	0
2022-03-04	164.490005	165.550003	162.100006	163.169998	83737200	0	0
2022-03-07	163.360001	165.020004	159.039993	159.300003	96418800	0	0
2022-03-08	158.820007	162.880005	155.800003	157.440002	131148300	0	0
2022-03-09	161.479996	163.410004	159.410004	162.949997	91454900	0	0
2022-03-10	160.199997	160.389999	155.979996	158.520004	105342000	0	0
2022-03-11	158.929993	159.279999	154.500000	154.729996	96836300	0	0
2022-03-14	151.449997	154.119995	150.100006	150.619995	108732100	0	0
2022-03-15	150.899994	155.570007	150.380005	155.089996	92964300	0	0
2022-03-16	157.050003	160.000000	154.460007	159.589996	101965749	0	0

With these features, we can use the framework preprocessing tools, add a few more technical indicators, and use it as a S_t in an MDP environment. Many data sources are already integrated into the framework, and the user can add customized ones. The data sources available are shown in table 3.

Technical Indicators

The OHLCV data is not enough on real trading algorithms, and other information needs to be considered. For example, it could be alternative data (news, balance sheet, investor's sentiment, and others) or technical indicators. These technical indicators could be momentum, moving average windows, change (in percent), Choppiness Indexes, and other information that tells how the stock prices change over time. As we saw in previous chapters, the MDP state can only contain information about the current S_t. These indicators are useful because they might

Table 3: List of supported data sources and their securities and frequency.

Data Source	Type	Range and Frequency
Alpaca	US Stocks, ETFs	2015-now, 1min
Baostock	CN Securities	1990-12-19-now, 5min
Binance	Cryptocurrency	API-specific, 1s, 1min
CCXT	Cryptocurrency	API-specific, 1min
IEXCloud	NMS US securities	1970-now, 1 day
JoinQuant	CN Securities	2005-now, 1min
QuantConnect	US Securities	1998-now, 1s
RiceQuant	CN Securities	2005-now, 1ms
tusharepro	CN Securities, A share	-now, 1 min
WRDS.TAQ	US Securities	2003-now, 1ms
Yahoo! Finance	US Securities	Frequency-specific, 1min

indicate trends without bringing information from the past (S_{t-n}) to current S_t. These indicators are provided by the *stockstats* (Zhuang, 2022) Python library. A complete list of all indicators is shown in Table 4.

5.1.3 Setting up the Environment

The environment design is one of the most important parts of RL solutions due to the nature of an MDP model. The task of automated portfolio optimization is interactive and stochastic. The training process involves observing stock price change, taking action, and calculating reward, and then, the agent will adjust its strategy. This will constantly happen, as the agent will interact with the environment and try to maximize the reward along the time. The environment on FinRL is built onto the OpenAI Gym framework (Brockman et al., 2016), and it simulates the stock market with discrete time-driven simulation. This makes the FinRL framework more flexible as it may use implementations of RL algorithms that already work with the Gym framework. The Gym framework also has some

Table 4: List of technical indicators available.

Statistics/Indicators	**Description**
middle	(close + high + low) / 3
compare	le, ge, lt, gt, eq, ne
count	both backward(c) and forward(fc)
cross	including upward cross and downward cross
SMA	Simple Moving Average
EMA	Exponential Moving Average
MSTD	Moving Standard Deviation
MVAR	Moving Variance
RSV	Raw Stochastic Value
RSI	Relative Strength Index
KDJ	Stochastic Oscillator
Bolling	Bollinger Band
MACD	Moving Average Convergence Divergence
CR	Energy Index (Intermediate Willingness Index)
WR	Williams Overbought/Oversold index
CCI	Commodity Channel Index
TR	True Range
ATR	Average True Range
DMA	Different of Moving Average (10, 50)
DMI	Directional Moving Index, including
+DI	Positive Directional Indicator
-DI	Negative Directional Indicator
ADX	Average Directional Movement Index
ADXR	Smoothed Moving Average of ADX
TRIX	Triple Exponential Moving Average
TEMA	Another Triple Exponential Moving Average
VR	Volume Variation Index
MFI	Money Flow Index
VWMA	Volume Weighted Moving Average
CHOP	Choppiness Index
KAMA	Kaufman's Adaptive Moving Average
PPO	Percentage Price Oscillator
StochRSI	Stochastic RSI
WT	LazyBear's Wave Trend
Supertrend	with the Upper Band and Lower Band

standards for initializing, resetting, stepping forward and has properties that let the agent know when the episode is terminated.

Action Space

The action space created by the FinRL varies by the number of stocks available and the number of shares that the agent is allowed to buy or sell. For example, a simple action space for a single stock would include three actions $A = \{-1, 0, 1\}$, representing selling, holding (or doing nothing), and buying, respectively, a single share. The environment provides a parameter k that can be used to define the number of shares allowed to buy or sell. That means that the action space range is defined as $A = \{-k, \cdots, -1, 0, 1, \cdots, k\}$. For example, if we set it to k=5, then the action space would be defined as $A = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$, where the number of actions is defined as $k \times 2 + 1$, for a single stock environment.

State Space

As the Gym environment is very customizable and the implemented algorithms are Gym ready, it is easy to create as many types of state spaces based on technical indicators, news sentiments, sector analysis, and others. This type of customization gives the power to do feature engineering even more robust and sophisticated.

5.1.4 Agents

In the Gym framework, the agents are responsible for observing the environment and performing the actions according to a policy. The FinRL environment comes with some agents already implemented out-of-the-box. For the algorithms, the already implemented agents use either ElegantRL, RLLib, or stablebaselines3. Table 5 shows the available agents and the typical use cases of their use.

Table 5: List of available algorithms and their input and output parameters.

Algorithms	Input	Output	Type	State-Action Space Support
DQN	States	Q-value	Value based	Discrete only
Double DQN	States	Q-value	Value based	Discrete only
Duelling DQN	States	Q-value	Value based	Discrete only
DDPG	States	Q-value	Actor-critic based	Continuous only
A2C	State action pair	Q-value	Actor-critic based	Discrete and continuous
PPO	State action pair	Q-value	Actor-critic based	Discrete and continuous
SAC	State action pair	Q-value	Actor-critic based	Continuous only
TD3	State action pair	Q-value	Actor-critic based	Continuous only
MADDPG	State action pair	Q-value	Actor-critic based	Continuous only

For this report's experiments, the A2C (Mnih et al., 2016), DDPG (Lillicrap et al., 2015), PPO (Schulman et al., 2017), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018) algorithms from the stablebaselines3 version of the FinRl framework.

This version is already compliant and optimized to be run with the format of the FinRL environments and agents.

5.1.5 Backtesting

Backtesting is the method for demonstrating how well a model would have performed using historical data. Backtesting is very similar to how data scientists test the models after training. The backtesting in FinRL is done with the pyfolio[1] library, which outputs a detailed performance and metrics and allows a more detailed comparison to the returns of a selected benchmark.

5.2 Dataset

This section explores the available datasets, particular features of each one of them, and some samples of the data to complete the description.

[1]https://github.com/quantopian/pyfolio

5.2.1 Market Data

The dataset used throughout this work corresponds to a historical series of data about the closing prices of the thirty stocks that are part of the DJIA.

The series represents 1784 days, from 31st December of 2012 to 31st January 2020, and were obtained using the embedded yfinance library in FinRL. Due to the data limitation, the dates boundaries were defined according to the Finsent dataset (see 5.2.2).

5.2.2 Finsent

The Finsent dataset is available only for sale and is published by the Nasdaq Data Link (formerly Quandl) and is created by InfoTrie.

The data set ranges from 31st December 2012 to 31st January 2020 (hence the limitations on 5.2.1) and is formed by daily news sentiment indicators for stocks derived from publicly available internet sources. Table 6 shows a preview of the format of the Finsent data available.

Table 6: Finsent sample data.

ticker	exchange_cd	date	sentiment	sentiment_high	sentiment_low	news_volume	news_buzz	name
TWTR	US	2018-06-01	0.858	0.858	-1.5	7	3	TWITTER
TWTR	US	2018-06-02	0	0	0	0	2	TWITTER
TWTR	US	2018-06-03	1	1	1	2	2	TWITTER
TWTR	US	2018-06-04	-2.571	-1	-3.4	7	3	TWITTER
TWTR	US	2018-06-05	0.422	4	0.089	57	10	TWITTER
TWTR	US	2018-06-06	-2.315	1	-4	19	4	TWITTER
TWTR	US	2018-06-07	0.358	2.125	-1.5	14	3	TWITTER

The date column does not contain the time part, which makes it impossible to determine at what time of the day this information would be available for the agent. The assumption is that the agent only makes the optimization at the end of the day, and the finsent information would be already known.

Table 7: Finsent data merged with market data.

date	open	high	low	close	volume	tic	day	sentiment	sentiment high	sentiment low	news volume	news buzz
2018-06-01 00:00:00	35.13999938964844	36.689998626708984	35.09000015258789	36.650001525878906	29583100	TWTR	4	0.858	0.858	-1.5	7.0	3.0
2018-06-04 00:00:00	36.45000076293945	37.979999542236328	35.95000076293945	37.880001068115234	32632800	TWTR	0	-2.571	-1.0	-3.4	7.0	3.0
2018-06-05 00:00:00	39.529998779296875	40.15999984741211	39.189998626708984	39.799999237060547	66122200	TWTR	1	0.422	4.0	0.089	57.0	10.0
2018-06-06 00:00:00	39.41999816894531	40.229999542236328	39.209999084472656	40.099998474121094	147805700	TWTR	2	-2.315	1.0	-4.0	19.0	4.0
2018-06-07 00:00:00	40.13999938964844	40.15999984741211	38.63999938964844	39.70000076293945	41573400	TWTR	3	0.358	2.125	-1.5	14.0	3.0

As shown in Table 7, before being used in the FinRL environment, the Finsent data is merged with the market data by the ticker name and date.

The data producer does not disclose what data sources are taken into account for building these daily data, nor the algorithms or steps used to score the sentiments or the news buzz.

Part III

Experiments

Chapter 6

Results

This chapter presents the experimental results performed to test the ability of the proposed DRL portfolio optimization using actual market data.

The experiments were conducted by training five DRL agents (A2C, DDPG, PPO, TD3, and SAC). Also, two types of datasets were used, one with only OHLCV data (plus the technical indicators) and another using the OHLCV and the finsent data (also, plus the technical indicators).

The split in the dataset considered for training began on 31st December 2012 and ended on 31st January 2019, while the dates for the backtesting were between 1st February 2019 and 1st February 2020. Dates that would include the beginning of the COVID-19 outbreak would be interesting for backtesting, but the Finsent data for those dates were not included on the dataset available. That leaves around seven years of data for training and one year for trading.

That was a concern while joining the Finsent data with the OHLCV data because as the Finsent had no time part on the date column, it might include future

information for the agent while performing the asset allocation at the end of the day. However, as the asset allocation only would happen at the closing auctions, it is assumed that all the news was already available and would be considered past information for the agent.

In the following figures, we present the results of backtesting all the algorithms trading in both environments, with and without the Finsent data. Algorithms pair the images to facilitate spotting the differences.

From Figure 4, it can be noted that for the TD3 agent, even though the cumulative returns stayed relatively the same, comparing the Finsent dataset with the regular, the returns received in 2019 (pre-pandemic announce) were greater the entire year, which made the mean of the annual return increase from 7.8% to 8.2%. This might be explained because the news score from December 2019 to March 2020 might not have reflected the actual situation of the markets, as the WHO declared COVID-19 as a pandemic around the second week of March, and the agent trading in the regular dataset did not have this information.

The drawdown chart represents the percentage loss of capital experienced by the portfolio before it starts to have positive returns, considering the daily closing prices. It shows that the TD3 agent had less continuous time in the underwater plot, but experienced higher peaks of losses. The peak of a drawdown is the highest drawdown that occurred during the period, and is used to assess the risk associated with the portfolio and the investment strategy.

For the DDPG agent, Figure 5 shows that the agent did not improve its cumulative returns while using the finsent dataset. The cumulative returns decreased and the mean Sharpe ratio went from 1% to around 0.5%, while the monthly returns improved significantly in January 2020. The underwater plot also reflects the

(a) Regular Dataset - TD3

(b) Finsent Dataset - TD3

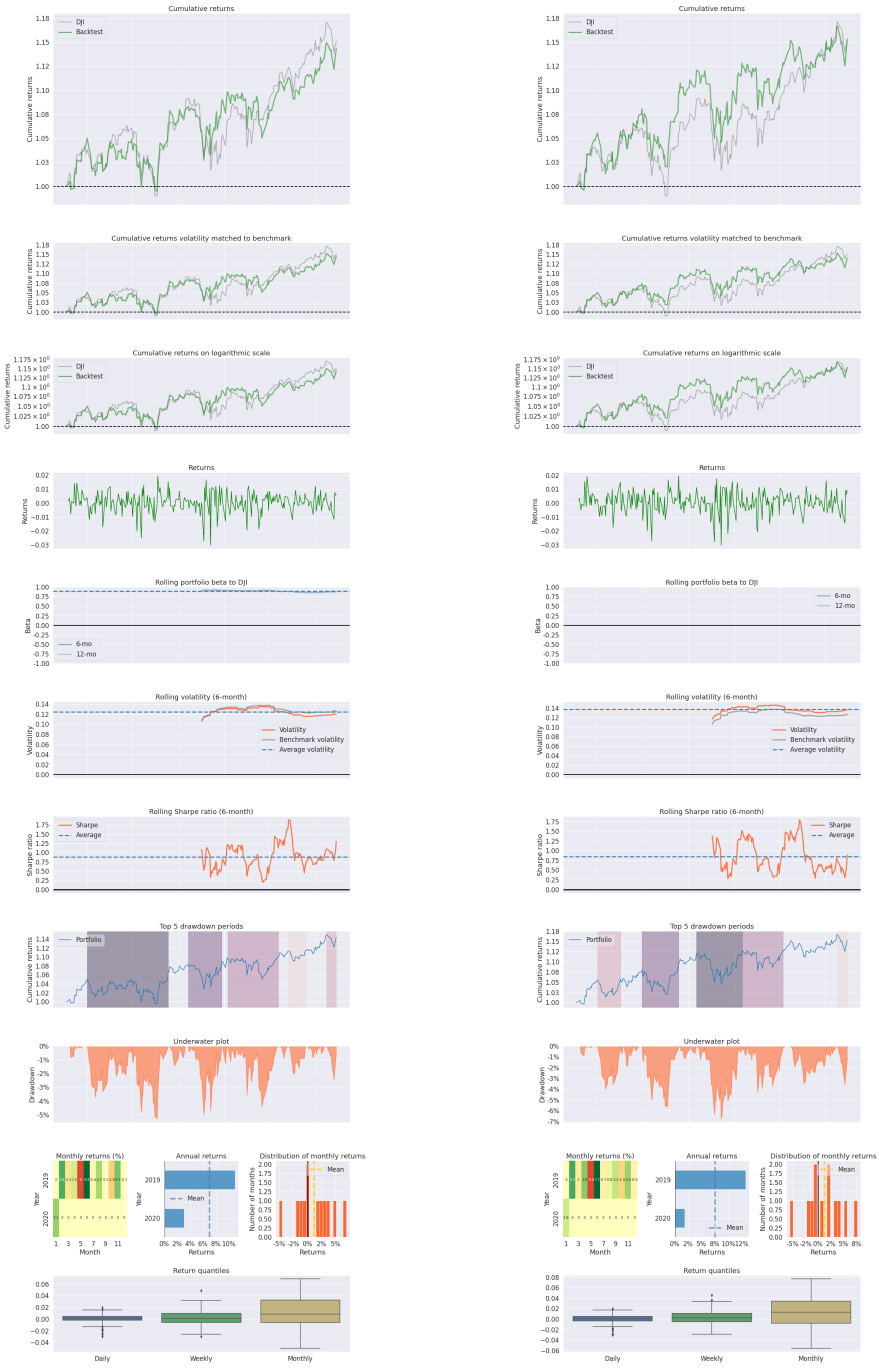

Figure 4: Comparison between the TD3 agent running with and without Finsent data.

(a) Regular Dataset - DDPG

(b) Finsent Dataset - DDPG

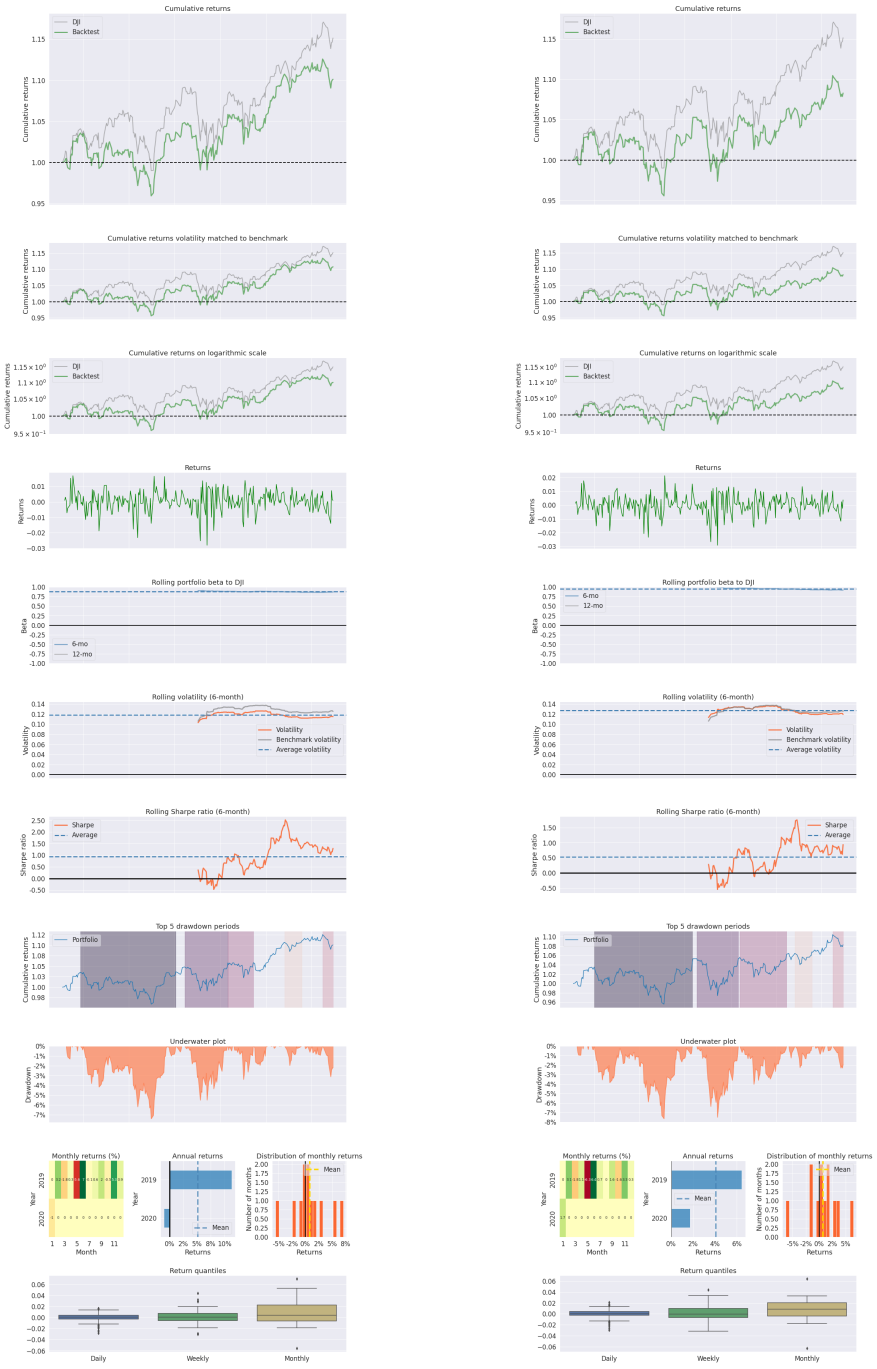

Figure 5: Comparison between DDPG agent running with and without Finsent data.

smaller and negative returns and shows that the portfolio remained continuously longer losing capital.

Figure 6 presents the results of the backtest for the PPO agent. It shows that, in general, the PPO agent had a slight improvement in cumulative returns and a good improvement in general performance while using the finsent data.

The underwater plot shows that even though it increased the time it stayed losing capital, the peak of the drawdown was smaller in the finsent dataset than on the regular dataset. Low drawdown peaks are preferred because they show that the losses from the investment were small.

The monthly return quantiles also show a concentration of positive returns compared to the quantiles of the regular dataset. Furthermore, the PPO agent improved in other technical indicators that showed a more stable trading strategy.

Figure 7 shows a clear trend of a decrease in the overall performance of the SAC agent. Although the agent had a good performance, beating the DJIA in almost all aspects when using the regular data, it lost to the index in the cumulative returns and indicators that show higher volatility of the risky portfolio strategy while using the finsent data.

The last comparison shown in Figure 8 presents a significant improvement in the trading strategy of the A2C agent. This portfolio using Finsent data had cumulative returns improve by 18% compared to the regular dataset. The other backtesting indicators also show that the cumulative returns are stable, making the portfolio less risky, even though the volatility increased. This fact is also sustained by the smaller peak drawdown and shorter continuous time in the underwater plot. Lastly, the Sharpe ratio gives a better idea of how the portfolio's returns perform against risk-free investments over the long term, showing that the portfolio has

(a) Regular Dataset - PPO

(b) Finsent Dataset - PPO

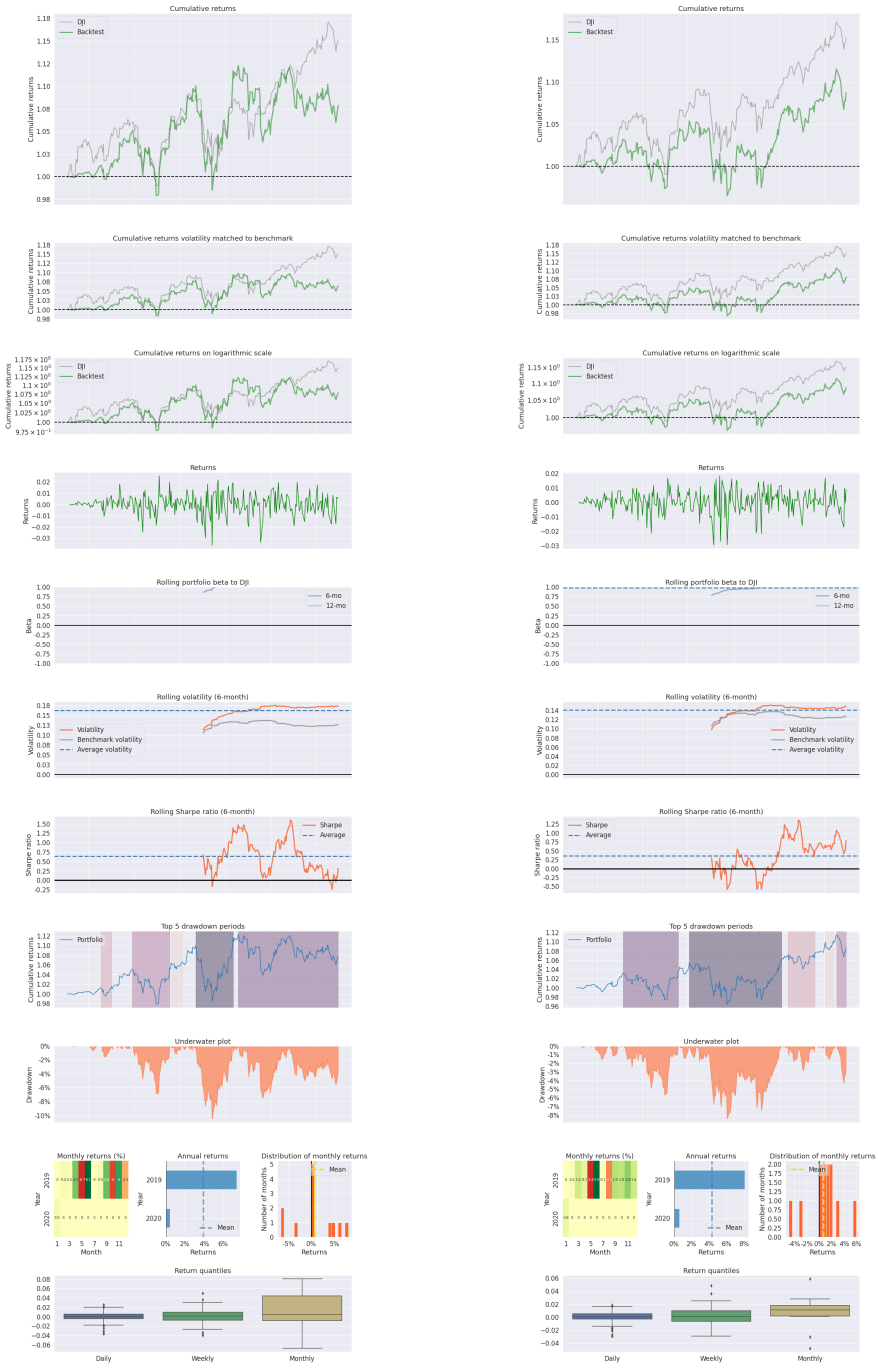

Figure 6: Comparison between the PPO agent running with and without Finsent data.

(a) Regular Dataset - SAC

(b) Finsent Dataset - SAC

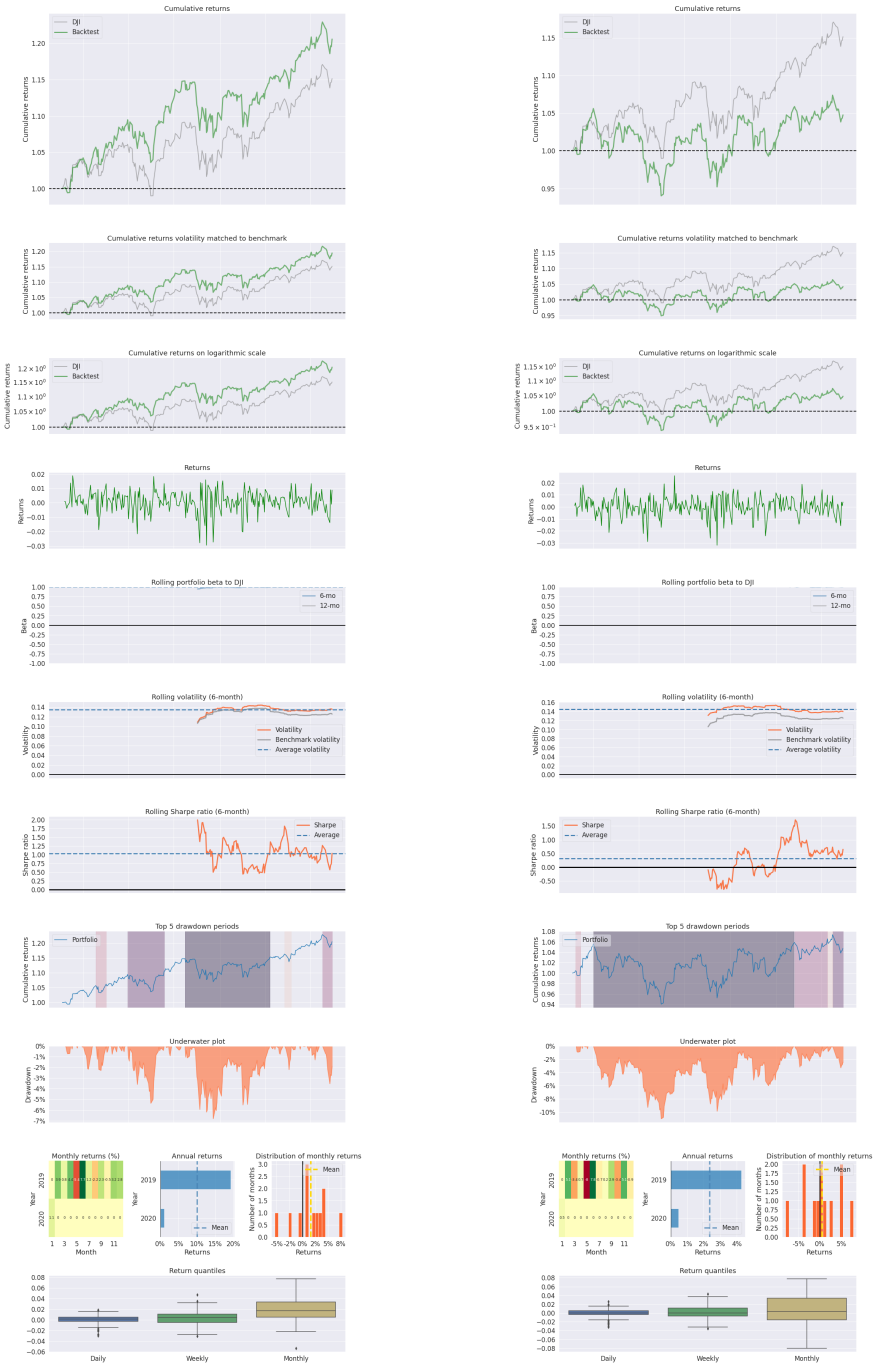

Figure 7: Comparison between the SAC agent running with and without Finsent data.

(a) Regular Dataset - A2C

(b) Finsent Dataset - A2C

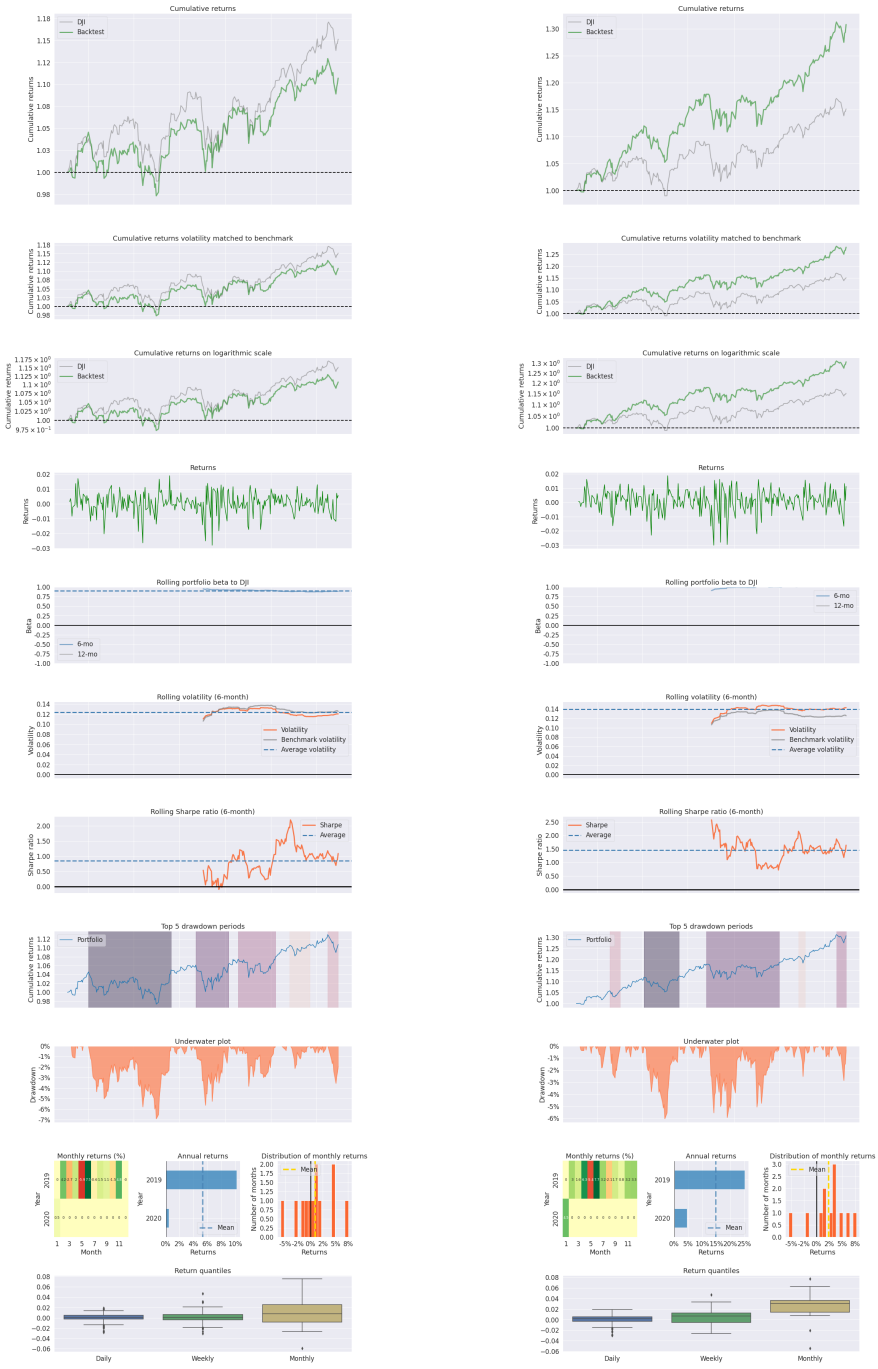

Figure 8: Comparison between A2C agent running with and without Finsent data.

sustainable returns and that the risk conditions are acceptable to good.

Chapter 7

Conclusion

The main goal of this report was to study the effectiveness of the Reinforcement Learning for Portfolio Optimization. To support this study, the concepts of domains such as ML and finance were explored and the FinRL framework was presented and extended to support Finsent data. In this chapter, the contributions, future work, and limitations of this research are explained.

7.1 Contributions

This thesis has made important contributions to the field. First, to build this report, we explored and tested different environments and agents' setups to use RL agents in the financial markets (as a stochastic discrete-time system). These setups led to many experiments to create a comparison of whether RL systems could be used for portfolio optimization and trading tasks at the same time.

Lastly, an extension of the environment class from the FinRL library was created to adapt the MDP state set to support the Finsent data. Within the limitations described in the next section, the agents trading in the regular dataset could

outperform the DJIA. Also, after using the finsent dataset, other trading agents could outperform the DJIA by a significant amount not only on the cumulative returns but also in other important aspects of the backtesting.

7.2 Limitations of the Research

The main limitations of this report are divided into categories: (i) Existing limitations on the used framework; (ii) Lack of alternative data open datasets; (iii) Chosen modeling architecture.

i The first is the limitation that is observed in the FinRL framework. The framework is still one of the most promising for applying RL algorithms to environments based on Gym, but it is still new, and it is somewhat unstable. In addition, some features that seem to be available in the framework documentation are not ready and deployed, which sometimes makes it confusing about the results. For example, the framework provides blueprints for environments for asset allocation, but it does not consider the transaction cost in this environment. Also, some unhandled exceptions might make the debugging of the environment complicated.

ii While it is relatively easy to find End of Day (EOD) closing prices for equities (even though real-time market data is expensive), it is not valid for alternative data. Alternative data is scarce and expensive for individuals. It would be interesting to join datasets of the prices with other datasets that might include exogenous data and see if it would improve the agents trading strategy. The Finsent data used in this study was bought for private use and, unfortunately, makes it difficult for others to replicate this work.

iii Lastly, the modeling chosen for this research limits the number of new assets that are added to the model. For this work, which had the DJIA benchmark, it is acceptable to have fixed thirty stocks for training and prediction, but unseen or new stocks (i.e. Initial Public Offerings (IPOs)) are not easily integrated. The model's network must be modified and retrained.

7.3 Future Work

Despite the overall performance of the trading agents on both the regular and Finsent datasets in the experiments, algorithms that use neural networks are black boxes. This lack of interpretability makes it difficult for investors to invest capital in a portfolio managed only by RL agents. Therefore, a good mix RL agents and human expert knowledge input could make it easier for the interpretability of the model actions and states.

Also, as pointed out in the limitation (iii), a modified version of the environment that would make it easier to add new assets without rebuilding and retraining the model would be closer to a solution that would be ready to use in the financial industry. For instance, Betancourt and Chen (2021) proposed a method for using RL for portfolio management with a dynamic number of assets, and it was tested successfully in the crypto market.

Another essential aspect that was not considered in the experiments undertaken in this report was hyperparameter tuning. All of the algorithms ran with the same parameters in both environments. However, it would be better to use different parameters, and the number of features from the simpler environment to the Finsent environment increases by about 40%.

Lastly, even though it is out of the scope of this report, an interesting avenue for

future research in this area is to connect the FinRL to a brokerage service and make the agents put the orders directly in the trading environment. This would require retraining so the agents would learn how to deal with some assumptions that we took out of the scope here: slippage, market impact, and sufficient liquidity.

Bibliography

Bailey, D. H. and López de Prado, M. (2013). An open-source implementation of the critical-line algorithm for portfolio optimization. *Algorithms*, 6(1):169–196.

Betancourt, C. and Chen, W.-H. (2021). Deep reinforcement learning for portfolio management of markets with a dynamic number of assets. *Expert Systems with Applications*, 164:114002.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI gym.

Callaway, E. (2020). 'it will change everything': DeepMind's AI makes gigantic leap in solving protein structures. *Nature*, 588(7837):203–204.

Chen, J. (1999). *Portfolio Investment*. `https://www.investopedia.com/terms/p/portfolio-investment.asp` [Accessed: 2022-03-15].

da Silva, R. F. (2021). *Automated stock trading system using deep reinforcement learning and price and sentiment prediction modules.* PhD thesis, Universidade de Sao Paulo.

de Prado, M. L. (2018). *Advances in Financial Machine Learning*. John Wiley & Sons Inc.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Jonathan Law, J. S. (2008). *A Dictionary of Finance and Banking*. OUP Oxford, 4th edition.

Kim, K.-j. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319. Support Vector Machines.

Kwak, Y., Song, J., and Lee, H. (2021). Neural network with fixed noise for index-tracking portfolio optimization. *Expert Systems with Applications*, 183:115298.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning.

Liu, X.-Y., Yang, H., Gao, J., and Wang, C. D. (2021). FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. *ACM International Conference on AI in Finance (ICAIF)*.

Martinez, L. C. and Pereira, A. C. M. (2020). Beating the stock market with a deep reinforcement learning day trading system. In *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*, pages 1–8. IEEE.

Meesad, P. and Rasel, R. I. (2013). Predicting stock market price using support vector regression. In *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 1–6.

Michaud, R. (1998). *Efficient asset management: a practical guide to stock portfolio optimization and asset allocation*. Oxford University Press.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Yun, H., Lee, M., Kang, Y. S., and Seok, J. (2020). Portfolio management via two-stage deep learning with a joint cost. *Expert Systems with Applications*, 143:113041.

Zhuang, C. (2022). stockstats. `https://github.com/jealous/stockstats` [Accessed: 2022-03-16].

Appendix A

FinRL Environment

A.1 Environment for Finsent Data

The current version of FinRL is not accepting custom indicators or data for the environment states, a modified version of the StockTradingEnv was created then. The code for this new version is shown on listing A.1.

```
import gym
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from gym import spaces
from gym.utils import seeding
from stable_baselines3.common.vec_env import DummyVecEnv
from typing import List
matplotlib.use("Agg")

class StockTradingEnv_Finsent(gym.Env):
    """A stock trading environment for OpenAI gym"""
```

```
14
15      metadata = {"render.modes": ["human"]}
16
17      def __init__(
18          self,
19          df: pd.DataFrame,
20          stock_dim: int,
21          hmax: int,
22          initial_amount: int,
23          num_stock_shares: List[int],
24          buy_cost_pct: List[float],
25          sell_cost_pct: List[float],
26          reward_scaling: float,
27          state_space: int,
28          action_space: int,
29          tech_indicator_list: List[str],
30          turbulence_threshold=None,
31          risk_indicator_col="turbulence",
32          make_plots: bool =False,
33          print_verbosity=10,
34          day=0,
35          initial=True,
36          previous_state=[],
37          model_name="",
38          mode="",
39          iteration="",
40          finsent_columns=['sentiment', 'sentiment_high', '
    sentiment_low', 'news_volume', 'news_buzz']
41      ):
42          self.day = day
43          self.df = df
44          self.stock_dim = stock_dim
45          self.hmax = hmax
```

```
46          self.num_stock_shares=num_stock_shares
47          self.initial_amount = initial_amount # get the initial
    cash
48          self.buy_cost_pct = buy_cost_pct
49          self.sell_cost_pct = sell_cost_pct
50          self.reward_scaling = reward_scaling
51          self.state_space = state_space
52          self.action_space = action_space
53          self.tech_indicator_list = tech_indicator_list
54          self.finsent_columns = finsent_columns
55          self.action_space = spaces.Box(low=-1, high=1, shape=(self
    .action_space,))
56          self.observation_space = spaces.Box(
57              low=-np.inf, high=np.inf, shape=(self.state_space,)
58          )
59          self.data = self.df.loc[self.day, :]
60          self.terminal = False
61          self.make_plots = make_plots
62          self.print_verbosity = print_verbosity
63          self.turbulence_threshold = turbulence_threshold
64          self.risk_indicator_col = risk_indicator_col
65          self.initial = initial
66          self.previous_state = previous_state
67          self.model_name = model_name
68          self.mode = mode
69          self.iteration = iteration
70          # initalize state
71          self.state = self._initiate_state()
72
73          # initialize reward
74          self.reward = 0
75          self.turbulence = 0
76          self.cost = 0
```

```
77        self.trades = 0
78        self.episode = 0
79        # memorize all the total balance change
80        self.asset_memory = [self.initial_amount+np.sum(np.array(
    self.num_stock_shares)*np.array(self.state[1:1+self.stock_dim])
    )] # the initial total asset is calculated by cash + sum (
    num_share_stock_i * price_stock_i)
81        self.rewards_memory = []
82        self.actions_memory = []
83        self.state_memory=[] # we need sometimes to preserve the
    state in the middle of trading process
84        self.date_memory = [self._get_date()]
85        #         self.logger = Logger('results',[CSVOutputFormat
    ])
86        # self.reset()
87        self._seed()
88
89     def _sell_stock(self, index, action):
90        def _do_sell_normal():
91            if self.state[index + 2*self.stock_dim + 1]!=True : #
    check if the stock is able to sell, for simlicity we just add
    it in techical index
92            # if self.state[index + 1] > 0: # if we use price<0 to
     denote a stock is unable to trade in that day, the total asset
     calculation may be wrong for the price is unreasonable
93                # Sell only if the price is > 0 (no missing data
    in this particular date)
94                # perform sell action based on the sign of the
    action
95                if self.state[index + self.stock_dim + 1] > 0:
96                    # Sell only if current asset is > 0
97                    sell_num_shares = min(
```

```
98                      abs(action), self.state[index + self.
    stock_dim + 1]
99                  )
100                 sell_amount = (
101                     self.state[index + 1]
102                     * sell_num_shares
103                     * (1 - self.sell_cost_pct[index])
104                 )
105                 # update balance
106                 self.state[0] += sell_amount
107
108                 self.state[index + self.stock_dim + 1] -=
    sell_num_shares
109                 self.cost += (
110                     self.state[index + 1] * sell_num_shares *
    self.sell_cost_pct[index]
111                 )
112                 self.trades += 1
113             else:
114                 sell_num_shares = 0
115         else:
116             sell_num_shares = 0
117
118         return sell_num_shares
119
120     # perform sell action based on the sign of the action
121     if self.turbulence_threshold is not None:
122         if self.turbulence >= self.turbulence_threshold:
123             if self.state[index + 1] > 0:
124                 # Sell only if the price is > 0 (no missing
    data in this particular date)
125                 # if turbulence goes over threshold, just
    clear out all positions
```

```
126                     if self.state[index + self.stock_dim + 1] > 0:
127                         # Sell only if current asset is > 0
128                         sell_num_shares = self.state[index + self.
    stock_dim + 1]
129                         sell_amount = (
130                             self.state[index + 1]
131                             * sell_num_shares
132                             * (1 - self.sell_cost_pct[index])
133                         )
134                         # update balance
135                         self.state[0] += sell_amount
136                         self.state[index + self.stock_dim + 1] = 0
137                         self.cost += (
138                             self.state[index + 1] *
    sell_num_shares * self.sell_cost_pct
139                         )
140                         self.trades += 1
141                     else:
142                         sell_num_shares = 0
143                 else:
144                     sell_num_shares = 0
145             else:
146                 sell_num_shares = _do_sell_normal()
147         else:
148             sell_num_shares = _do_sell_normal()
149 
150         return sell_num_shares
151 
152     def _buy_stock(self, index, action):
153         def _do_buy():
154             if self.state[index + 2*self.stock_dim+ 1] !=True: #
    check if the stock is able to buy
155             # if self.state[index + 1] >0:
```

```
156                 # Buy only if the price is > 0 (no missing data in
    this particular date)
157                 available_amount = self.state[0] / (self.state[
    index + 1]*(1 + self.buy_cost_pct[index])) # when buying stocks
    , we should consider the cost of trading when calculating
    available_amount, or we may be have cash<0
158                 # print('available_amount:{}'.format(
    available_amount))
159
160                 # update balance
161                 buy_num_shares = min(available_amount, action)
162                 buy_amount = (
163                     self.state[index + 1] * buy_num_shares * (1 +
    self.buy_cost_pct[index])
164                 )
165                 self.state[0] -= buy_amount
166
167                 self.state[index + self.stock_dim + 1] +=
    buy_num_shares
168
169                 self.cost += self.state[index + 1] *
    buy_num_shares * self.buy_cost_pct[index]
170                 self.trades += 1
171             else:
172                 buy_num_shares = 0
173
174             return buy_num_shares
175
176         # perform buy action based on the sign of the action
177         if self.turbulence_threshold is None:
178             buy_num_shares = _do_buy()
179         else:
180             if self.turbulence < self.turbulence_threshold:
```

```
181                 buy_num_shares = _do_buy()
182             else:
183                 buy_num_shares = 0
184                 pass
185 
186         return buy_num_shares
187 
188     def _make_plot(self):
189         plt.plot(self.asset_memory, "r")
190         plt.savefig("results/account_value_trade_{}.png".format(
    self.episode))
191         plt.close()
192 
193     def step(self, actions):
194         self.terminal = self.day >= len(self.df.index.unique()) -
    1
195         if self.terminal:
196             # print(f"Episode: {self.episode}")
197             if self.make_plots:
198                 self._make_plot()
199             end_total_asset = self.state[0] + sum(
200                 np.array(self.state[1 : (self.stock_dim + 1)])
201                 * np.array(self.state[(self.stock_dim + 1) : (self
    .stock_dim * 2 + 1)])
202             )
203             df_total_value = pd.DataFrame(self.asset_memory)
204             tot_reward = (
205                 self.state[0]
206                 + sum(
207                     np.array(self.state[1 : (self.stock_dim + 1)])
208                     * np.array(
209                         self.state[(self.stock_dim + 1) : (self.
    stock_dim * 2 + 1)]
```

```
                    )
                )
                - self.asset_memory[0]
            ) # initial_amount is only cash part of our initial asset
            df_total_value.columns = ["account_value"]
            df_total_value["date"] = self.date_memory
            df_total_value["daily_return"] = df_total_value["account_value"].pct_change(
                1
            )
            if df_total_value["daily_return"].std() != 0:
                sharpe = (
                    (252 ** 0.5)
                    * df_total_value["daily_return"].mean()
                    / df_total_value["daily_return"].std()
                )
            df_rewards = pd.DataFrame(self.rewards_memory)
            df_rewards.columns = ["account_rewards"]
            df_rewards["date"] = self.date_memory[:-1]
            if self.episode % self.print_verbosity == 0:
                print(f"day: {self.day}, episode: {self.episode}")
                print(f"begin_total_asset: {self.asset_memory[0]:0.2f}")
                print(f"end_total_asset: {end_total_asset:0.2f}")
                print(f"total_reward: {tot_reward:0.2f}")
                print(f"total_cost: {self.cost:0.2f}")
                print(f"total_trades: {self.trades}")
                if df_total_value["daily_return"].std() != 0:
                    print(f"Sharpe: {sharpe:0.3f}")
                print("=================================")

            if (self.model_name != "") and (self.mode != ""):
```

```
240                 df_actions = self.save_action_memory()
241                 df_actions.to_csv(
242                     "results/actions_{}_{}_{}.csv".format(
243                         self.mode, self.model_name, self.iteration
244                     )
245                 )
246                 df_total_value.to_csv(
247                     "results/account_value_{}_{}_{}.csv".format(
248                         self.mode, self.model_name, self.iteration
249                     ),
250                     index=False,
251                 )
252                 df_rewards.to_csv(
253                     "results/account_rewards_{}_{}_{}.csv".format(
254                         self.mode, self.model_name, self.iteration
255                     ),
256                     index=False,
257                 )
258                 plt.plot(self.asset_memory, "r")
259                 plt.savefig(
260                     "results/account_value_{}_{}_{}.png".format(
261                         self.mode, self.model_name, self.iteration
262                     ),
263                     index=False,
264                 )
265                 plt.close()
266 
267             # Add outputs to logger interface
268             # logger.record("environment/portfolio_value",
    end_total_asset)
269             # logger.record("environment/total_reward", tot_reward
    )
```

```
270             # logger.record("environment/total_reward_pct", (
    tot_reward / (end_total_asset - tot_reward)) * 100)
271             # logger.record("environment/total_cost", self.cost)
272             # logger.record("environment/total_trades", self.
    trades)
273 
274             return self.state, self.reward, self.terminal, {}
275 
276         else:
277             actions = actions * self.hmax  # actions initially is
    scaled between 0 to 1
278             actions = actions.astype(
279                 int
280             )  # convert into integer because we can't by fraction
     of shares
281             if self.turbulence_threshold is not None:
282                 if self.turbulence >= self.turbulence_threshold:
283                     actions = np.array([-self.hmax] * self.
    stock_dim)
284             begin_total_asset = self.state[0] + sum(
285                 np.array(self.state[1 : (self.stock_dim + 1)])
286                 * np.array(self.state[(self.stock_dim + 1) : (self
    .stock_dim * 2 + 1)])
287             )
288             # print("begin_total_asset:{}".format(
    begin_total_asset))
289 
290             argsort_actions = np.argsort(actions)
291             sell_index = argsort_actions[: np.where(actions < 0)
    [0].shape[0]]
292             buy_index = argsort_actions[::-1][: np.where(actions >
     0)[0].shape[0]]
293 
```

```
294             for index in sell_index:
295                 # print(f"Num shares before: {self.state[index+
    self.stock_dim+1]}")
296                 # print(f'take sell action before : {actions[index
    ]}')
297                 actions[index] = self._sell_stock(index, actions[
    index]) * (-1)
298                 # print(f'take sell action after : {actions[index
    ]}')
299                 # print(f"Num shares after: {self.state[index+self
    .stock_dim+1]}")
300
301             for index in buy_index:
302                 # print('take buy action: {}'.format(actions[index
    ]))
303                 actions[index] = self._buy_stock(index, actions[
    index])
304
305             self.actions_memory.append(actions)
306
307             # state: s -> s+1
308             self.day += 1
309             self.data = self.df.loc[self.day, :]
310             if self.turbulence_threshold is not None:
311                 if len(self.df.tic.unique()) == 1:
312                     self.turbulence = self.data[self.
    risk_indicator_col]
313                 elif len(self.df.tic.unique()) > 1:
314                     self.turbulence = self.data[self.
    risk_indicator_col].values[0]
315             self.state = self._update_state()
316
317             end_total_asset = self.state[0] + sum(
```

```
                np.array(self.state[1 : (self.stock_dim + 1)])
                * np.array(self.state[(self.stock_dim + 1) : (self
.stock_dim * 2 + 1)])
            )
            self.asset_memory.append(end_total_asset)
            self.date_memory.append(self._get_date())
            self.reward = end_total_asset - begin_total_asset
            self.rewards_memory.append(self.reward)
            self.reward = self.reward * self.reward_scaling
            self.state_memory.append(self.state) # add current
state in state_recorder for each step

        return self.state, self.reward, self.terminal, {}

    def reset(self):
        # initiate state
        self.state = self._initiate_state()

        if self.initial:
            self.asset_memory = [self.initial_amount+np.sum(np.
array(self.num_stock_shares)*np.array(self.state[1:1+self.
stock_dim]))]
        else:
            previous_total_asset = self.previous_state[0] + sum(
                np.array(self.state[1 : (self.stock_dim + 1)])
                * np.array(
                    self.previous_state[(self.stock_dim + 1) : (
self.stock_dim * 2 + 1)]
                )
            )
            self.asset_memory = [previous_total_asset]

        self.day = 0
```

```
346         self.data = self.df.loc[self.day, :]
347         self.turbulence = 0
348         self.cost = 0
349         self.trades = 0
350         self.terminal = False
351         # self.iteration=self.iteration
352         self.rewards_memory = []
353         self.actions_memory = []
354         self.date_memory = [self._get_date()]
355 
356         self.episode += 1
357 
358         return self.state
359 
360     def render(self, mode="human", close=False):
361         return self.state
362 
363     def _initiate_state(self):
364         if self.initial:
365             # For Initial State
366             if len(self.df.tic.unique()) > 1:
367                 # for multiple stock
368                 state = (
369                     [self.initial_amount]
370                     + self.data.close.values.tolist()
371                     + self.num_stock_shares
372                     + sum(
373                         [
374                             self.data[tech].values.tolist()
375                             for tech in self.tech_indicator_list
376                         ],
377                         [],
378                     )
```

```
                    + sum(
                        [
                            self.data[finsent].values.tolist()
                            for finsent in self.finsent_columns
                        ],
                        [],
                    )
                ) # append initial stocks_share to initial state,
    instead of all zero
            else:
                # for single stock
                state = (
                    [self.initial_amount]
                    + [self.data.close]
                    + [0] * self.stock_dim
                    + sum([[self.data[tech]] for tech in self.
    tech_indicator_list], [])
                )
        else:
            # Using Previous State
            if len(self.df.tic.unique()) > 1:
                # for multiple stock
                state = (
                    [self.previous_state[0]]
                    + self.data.close.values.tolist()
                    + self.previous_state[
                        (self.stock_dim + 1) : (self.stock_dim * 2
     + 1)
                    ]
                    + sum(
                        [
                            self.data[tech].values.tolist()
                            for tech in self.tech_indicator_list
```

```
                        ],
                        [],
                    )
                    + sum(

                        [
                            self.data[finsent].values.tolist()
                            for finsent in self.finsent_columns
                        ],
                        [],
                    )
                )
            else:
                # for single stock
                state = (
                    [self.previous_state[0]]
                    + [self.data.close]
                    + self.previous_state[
                        (self.stock_dim + 1) : (self.stock_dim * 2
 + 1)
                    ]
                    + sum([[self.data[tech]] for tech in self.
tech_indicator_list], [])
                )
        return state

    def _update_state(self):
        if len(self.df.tic.unique()) > 1:
            # for multiple stock
            state = (
                [self.state[0]]
                + self.data.close.values.tolist()
```

```
439                 + list(self.state[(self.stock_dim + 1) : (self.
    stock_dim * 2 + 1)])
440                 + sum(
441                     [
442                         self.data[tech].values.tolist()
443                         for tech in self.tech_indicator_list
444                     ],
445                     [],
446                 )
447                 + sum(
448                     [
449                         self.data[finsent].values.tolist()
450                         for finsent in self.finsent_columns
451                     ],
452                     [],
453                 )
454             )
455 
456         else:
457             # for single stock
458             state = (
459                 [self.state[0]]
460                 + [self.data.close]
461                 + list(self.state[(self.stock_dim + 1) : (self.
    stock_dim * 2 + 1)])
462                 + sum([[self.data[tech]] for tech in self.
    tech_indicator_list], [])
463             )
464 
465         return state
466 
467     def _get_date(self):
468         if len(self.df.tic.unique()) > 1:
```

```
469             date = self.data.date.unique()[0]
470         else:
471             date = self.data.date
472         return date
473 
474     # add save_state_memory to preserve state in the trading
    process
475     def save_state_memory(self):
476         if len(self.df.tic.unique()) > 1:
477             # date and close price length must match actions
    length
478             date_list = self.date_memory[:-1]
479             df_date = pd.DataFrame(date_list)
480             df_date.columns = ["date"]
481 
482             state_list = self.state_memory
483             df_states = pd.DataFrame(state_list,columns=['cash','
    Bitcoin_price','Gold_price','Bitcoin_num','Gold_num','
    Bitcoin_Disable','Gold_Disable'])
484             df_states.index = df_date.date
485             # df_actions = pd.DataFrame({'date':date_list,'actions
    ':action_list})
486         else:
487             date_list = self.date_memory[:-1]
488             state_list = self.state_memory
489             df_states = pd.DataFrame({"date": date_list, "states":
     state_list})
490         # print(df_states)
491         return df_states
492 
493     def save_asset_memory(self):
494         date_list = self.date_memory
495         asset_list = self.asset_memory
```

```
496         # print(len(date_list))
497         # print(len(asset_list))
498         df_account_value = pd.DataFrame(
499             {"date": date_list, "account_value": asset_list}
500         )
501         return df_account_value
502 
503     def save_action_memory(self):
504         if len(self.df.tic.unique()) > 1:
505             # date and close price length must match actions
    length
506             date_list = self.date_memory[:-1]
507             df_date = pd.DataFrame(date_list)
508             df_date.columns = ["date"]
509 
510             action_list = self.actions_memory
511             df_actions = pd.DataFrame(action_list)
512             df_actions.columns = self.data.tic.values
513             df_actions.index = df_date.date
514             # df_actions = pd.DataFrame({'date':date_list,'actions
    ':action_list})
515         else:
516             date_list = self.date_memory[:-1]
517             action_list = self.actions_memory
518             df_actions = pd.DataFrame({"date": date_list, "actions
    ": action_list})
519         return df_actions
520 
521     def _seed(self, seed=None):
522         self.np_random, seed = seeding.np_random(seed)
523         return [seed]
524 
525     def get_sb_env(self):
```

```
526         e = DummyVecEnv([lambda: self])
527         obs = e.reset()
528         return e, obs
```

Listing A.1: StockTradingEnv compatible with Finsent information.