Deep Reinforcement Learning for Stock Portfolio Allocation with Sentiment Analysis and Technical Indicators

Oscar Wignall

Master of Science in Computer Science
University of Bath
September 2021

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Deep Reinforcement Learning for Stock Portfolio Allocation with Sentiment Analysis and Technical Indicators

Submitted by: Oscar Wignall

Copyright

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see `https://www.bath.ac.uk/publications/university-ordinances/attachments/Ordinances_1_October_2020.pdf`).
This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Algorithmic trading of stocks provides the opportunity to outperform human traders in a number of ways. Deep reinforcement learning as an approach to this problem is an active area of research. In this project a portfolio allocation environment for deep reinforcement learning is developed. This is then used to conduct a series of experiments to learn more about the effectiveness of specific state features, such as technical indicators and sentiment derived features, and the effectiveness of different deep reinforcement learning algorithms, specifically critic-only and actor-critic algorithms. As part of the software, data extraction, processing and storage were also included. For the experimentation 3 Hypotheses were proposed, which were then tested with 5 Experiments. The results of these experiments provided statistically significant evidence that. 1. Agents using either sentiment analysis features or technical indicators produce higher returns than untrained agents in a portfolio allocation environment. 2. A2C agents produce higher returns than DQN agents in a portfolio allocation environment. A limitation of modelling the environment as portfolio allocation is that transaction costs are not accounted for, which limits the real world applicability of the approach. An effective portfolio allocation environment could however still be used as a component of a more complete trading strategy.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my personal supervisors, James Laird and George Fletcher, for their help and support throughout the process of this project. I would also like to thank my former colleagues at 4 Shires Asset Management for all I learnt about investment and the financial markets during my time there. I am very grateful to have had the opportunity to pursue this project and it has been a very fulfilling experience.

Chapter 1

Introduction

1.1 Background

1.1.1 Description of Problem

Forecasting movements in the stock market is an area of great importance. Even minor improvements to the trading strategy of a financial institution with large capital holdings has the potential to be very financially beneficial. Algorithmic trading is an approach to trading financial assets which executes trades without any human interference. This can provide value in a number of ways including:

- Faster response to events.
- Reduced behavioural biases and human error.
- Ability to act on more information than an individual could reasonably consume.

In this project I will seek to address this problem by building on, and attempting to make improvements, to existing work. Specifically, I will address this problem by building on the knowledge in the area of deep reinforcement learning with sentiment analysis.

1.1.2 Information Used for Prediction

Information with which to make decisions is an important consideration. The usefulness of data will depend on both its availability and its ease of interpretation.

Historical price and volumes data can be useful for making predictions about the future value of stocks. Many metrics can be derived from historical price data such as volatility, moving averages and many more technical indicators. This information can be interpreted directly by traders or can be used by an algorithmic trading strategy. The availability of this data is high as prices are constantly updating for stocks.

Financial data on a company includes things such as its revenue, income, and net assets. This information can be used by investors to produce a more intrinsic valuation of a company. Publicly traded companies are required to produce financial statements each year as well as interim financial statements. This data is therefore easily available but less frequently updated. The way in which companies record their accounts will vary and certain figures may be significantly more or less important depending on the specific industry and company.

Textual data such as news articles or social media posts. This data provides useful information in two ways. Firstly, it can provide information that could shed light on the future intrinsic value of the stock. Secondly, it can provide an insight into the sentiment of traders towards the stock. This is important because ultimately the value of stocks is determined by the opinion of market participants. Although there is a great deal of this data available its interpretation is complex. Individuals reading news can interpret the article and draw conclusions about what it means for a stock. Drawing conclusions from this sort of data however is more difficult for a computer. On the other hand however a computer has the ability to quickly go through huge quantities of textual data very quickly, but an individual has significant limitations on the speed at which they can consume this information.

Investment ratings by financial institutions are another potentially useful source of information which also provides sentiment towards stocks that can be interpreted with relative ease by either human traders or algorithmic approaches.

Even with all this information it is still very difficult to accurately predict the movements of stock prices. One of the main reasons for this is that a great deal of stock prices movement comes as a result of events that could not have been anticipated with this limited information. There is also the simple fact that there is some randomness associated with investor behaviour.

1.1.3 Approaches Overview

In this project the focus will be on the use of deep reinforcement learning for stock price prediction; with the state representation of this model derived from price and volumes data as well as sentiment sources. This decision was informed by research into deep learning methods and sentiment analysis.

Hu, Zhao and Khushi (2021) provides a survey of the use of deep learning for price prediction of stocks and forex. This survey divides papers into the following categories. CNN, LSTM, DNN, RNN, Reinforcement Learning, and Other Deep Learning Methods. The survey charted the methods by number of papers used in the survey and LSTM was the most used. Due to the nature of the problem however there is some crossover where papers have combined methods. There were a number of papers in the reinforcement learning section of interest. A popular approach by some of these papers, such as Li, Ni and Chang (2019b); Jia et al. (2019), was to use deep reinforcement learning with LSTM for function approximation. This is useful as it utilizes LSTMs ability to handle time series data. Shin, Ra and Choi (2019) is another interesting paper that uses deep reinforcement learning. This paper takes the approach of generating charts from various forms of technical data and then using these as inputs to a CNN layer. The results of which in turn are inputted into a LSTM layer. This is a unique approach to identification of technical features. However, these papers did not make use of sentiment data as an input to their models.

Another survey by Obthong et al. (2020) outlines some of the most popular machine learning approaches to stock price prediction. This survey looks at a broad range of papers and provides more information specifically on papers using sentiment analysis. The survey identifies financial news and social media as the two main sources of textual data used by papers. These sources differ in that news has far more of a reliance on facts whereas social media is very subjective and more likely based on rumours, and therefore more likely to be inaccurate (Obthong et al., 2020). One interesting paper mentioned by the survey that uses news is Nam and Seong (2019). This paper incorporates not just news on the specific stock, but also news on stocks

within the same sector that are likely to have a causal effect. Implementing this approach utilizes more available data and outperformed the single stock news approach. Shah, Isah and Zulkernine (2018) is another paper mentioned which highlights the importance of the specific industry selected. For instance, the pharmaceutical sector is very sensitive to certain news data because this news may include information on the approval of the companies drugs. This can have a very significant affect on the intrinsic value of the company.

Nan, Perumal and Zaiane (2020) is a paper which makes use of both sentiment and deep reinforcement learning in its approach, making it an important resource for this project. The paper uses average sentiment towards a company as part of the state space in its MDP by reducing sentiment data to a single value. This paper also employs an interesting approach to collection of news. Instead of only including news that directly mentions the stock or company, google knowledge graph is used to identify related topics and keywords. Koratamaddi et al. (2021) is another paper using sentiment analysis and reinforcement learning but this paper uses an actor-critic model instead of a critic-only one.

1.1.4 Resources Overview

For data extraction there are a number of useful libraries that can be used. Requests, BeautifulSoup, and Selenium can all be used to web scrape in Python. For stock price prediction modules such as yfinance can also be useful as a way to get stock data directly.

In terms of data processing and analysis there are also many useful tools. For sentiment analysis tools such as Textblob, VaderSentiment, IBM tone analyzer and HuggingFace can all be used to extract sentiment from text. For deep reinforcement learning function approximation, neural networks can be implemented with libraries such as Tensorflow and PyTorch. Or alternatively a more complete tool could be used for the model such as Stable baselines 3.

1.2 Objectives and Deliverables

The overall objective of this project will be to develop and compare unique agents that use deep reinforcement learning and sentiment analysis to make trading actions with stock data. These models will be compared against each other and untrained agents.

1.2.1 Objectives

The main objectives for this project are as follows.

1. Perform a comprehensive review of the relevant literature and technology.
2. Develop software for construction of portfolio allocation environments that use both technical indicators and sentiment features.
3. Compare the performance of agents using different features individually and together when used in a portfolio allocation environment.
4. Compare the performance of critic-only and actor-critic agents in a portfolio allocation environment.

The first objective of this project is producing a comprehensive literature and technology review, which will inform design decisions for the rest of the project and put the contributions into context.

Objective 2 is to develop software for the construction of portfolio allocation environments for reinforcement learning. This software should be able to extract, process and store data and then use this data for environment construction.

Objectives 3 and 4 will be achieved by first constructing the relevant agents and then comparing them in terms of performance with a number of experiments.

1.2.2 Deliverables

By the completion of the project I hope to have produced the following deliverables.

- Literature and technology review of relevant work.
- Software for obtaining necessary technical data on a given stock.
- Software for extracting news articles for a given stock over a given time range.
- Software for sentiment analysis of given news articles.
- Actor-critic deep reinforcement learning models for technical indicators and sentiment analysis individually.
- Actor-critic deep reinforcement learning model using both of the inputs together.
- Critic-only deep reinforcement learning model using all inputs.
- Experimental results comparing all of the actor-critic models against each other and the market portfolio.
- Experimental results comparing the critic-only and actor-critic models.

1.3 Structure

This paper will start off with the literature and technology review, which will consist of three parts. Firstly, the stock price prediction section will look at some of the common information and approaches used in this area; as well as the ways in which these approaches are evaluated. Next, the sentiment analysis section will look at sentiment analysis both generally and in the context of stock price prediction. After this the review will look at deep reinforcement learning, covering important concepts and relevant work in this area. The last part of the review will look at some relevant resources that are available, focusing on those that may be appropriate for use in this project. The next chapter of the paper will be the methodology where the approach towards software development and experimentation will be covered. The implementation chapter will then proceed to document and explain the development of the software. Following the implementation the experimentation chapter will cover the approach to the individual experiments and discuss the results produced by them. Finally, the conclusion will summarize the paper, give a critical evaluation of the outcome, present the contributions, and suggest potential further work.

Chapter 2

Stock Price Prediction

2.1 Technical and Fundamental Indicators

2.1.1 Overview and Comparison

In stock analysis there are generally two main approaches. These are Fundamental analysis and Technical analysis. Fundamental analysis focuses on trying to identify what the intrinsic value of a stock is compared to its price. Technical analysis on the other hand looks at the stock in terms of price and volume movements in the hopes of identifying features such as trends and momentum.

Both of these approaches consist of their own unique metrics and indicators for use in analysis. The value of these metrics in the context of this project is the ability to derive numerical values which can make up the state perceived by a reinforcement learning agent. We could of course simply use objective stock data, such as price and volume. However, using indicators which are derived from this objective data simplifies the problem for our reinforcement learning value approximation function.

Beyaz et al. (2018) provides a valuable comparison of these two types of indicator in the context of machine learning and find both are individually useful for stock price prediction models and that an approach of combining the two is the most effective approach. Works such as Bettman, Sault and Schultz (2009) also highlight their complementary nature in a more traditional linear prediction framework.

2.1.2 Technical Indicators

Technical analysis is a broad area and there are many different indicators available in the area of stock price prediction. The majority of technical indicators are derived from price and volume data and as such can be continuously updated with market movements. In the context of machine learning these indicators have been both popular and effective (Oriani and Coelho, 2016; Larsen, 2010). Rockefeller (2019) provides a comprehensive overview of the area of technical analysis and gives detailed descriptions of many of the most popular indicators. Table 2.1 lists some of the most significant of these in the area of machine learning for stock price prediction. Figure 2.1 is a chart with some of these indicators plotted alongside price.

Indicator	Parameters	Description
SMA	Lengths	Simple moving average of prices within set range.
EMA	Lengths, Smoothing	Exponential moving average of prices within set range.
MACD	Long-length, Short-length	Measures difference between Short EMA and Long EMA.
RSI	Length	Relative strength index is a momentum oscillator.
Stochastic Oscillator	Length	Indicator for generating overbought and oversold signals.
Bollinger Bands	Length, Standard deviations	Three line indicator for generating overbought and oversold signals.
ROC	-	Simply the percentage change in the price.
OBV	-	Momentum indicator which uses volume changes to make price predictions.

Table 2.1: Technical indicators

Figure 2.1: Chart of technical indicators

2.1.3 Fundamental Indicators

Fundamental indicators, unlike technical indicators, are derived partially from information on the actual finances of a company. The benefit of these metrics is that they can help to approximate the actual intrinsic value of a stock (Krantz, 2016). The limitation however comes from the fact that financial data is only released quarterly. In the context of machine learning the need for large data sets has meant that fundamental indicators have received less attention (Beyaz et al., 2018). To get around this issue we can restrict our fundamental indicators to those which include price and as such update with the same frequency as technical indicators.

Table 2.2 provides some of the most popular fundamental indicators, specifically those that are suited to a machine learning model.

Indicator	Formula	Description
Trailing PE	$PE_t = P/E_t$	Price divided by most recent earnings
Forward PE	$PE_f = P/E_f$	Price divided by forecast for coming earnings
PEG	$PEG = (P/E)/G$	Price earnings ratio divided by the annual earnings growth rate
Price-To-Book	$PB = P/B$	Price divided by the book value per share
Debt-To-Equity	$DE = D/EQ$	Total debt divided by total equity where $EQ = P \times N$ and N is the number of shares outstanding
Dividend Yield	$DY = DPS/P$	Annual dividends per share divided by price

Table 2.2: Fundamental indicators

2.2 Evaluation Approaches

2.2.1 Statistical Approach

For stock price prediction there are generally two main approaches to modelling the environment and evaluating the performance. The first is to work towards a model that is evaluated based on its correlation with the actual changes in price of a stock. This can be done by comparing model predictions with the actual prices and calculating the mean squared error. This approach is popular with linear prediction models and supervised machine learning models.

2.2.2 Algorithmic Trading Approach

The other approach is for the agent to simulate actually trading stocks with the intention of generating return. This is more closely aligned with algorithmic trading and involves managing an investment portfolio of stocks. This method is popular with deep reinforcement learning approaches for stock price prediction as it frames the problem as a game. It is also arguably a more important area for research as it is closely aligned with the real world problem of portfolio management.

This approach is similar in principle to the idea of backtesting an investment strategy, as described by Pedersen (2015), which involves selecting a universe of available stocks and information which can be used to inform trading decisions. It is important to only include information that was available at the time when the action would have needed to have been made. For instance, the model should not be using a companies financial data from 2017 for a decision in Jan 2018 if this financial information did not actually become publicly available until Feb 2018. Performance of the model can then be measured in terms of the return generated by the trading strategy compared to a benchmark. This could be a relevant investment index, an equally weighted portfolio of all stocks in the model universe, an untrained model, or a random strategy.

An investment strategy however should be measured not just by generated return but also by the associated risk. A useful metric for this is the Sharpe ratio which considers both risk and return. The equation for this is given in Equation 2.1, where R_a is the actual return generated, R_f is the risk free rate of return, and σ_a is the standard deviation of the returns. The risk free rate is generally given as the return an investor could get from a fixed rate government bond. This is the most popular metric used for the inclusion of risk (Koratamaddi et al., 2021; Nan, Perumal and Zaiane, 2020).

$$S_a = \frac{R_a - R_f}{\sigma_a} \tag{2.1}$$

There are other ways of evaluating the quality of an investment strategy, such as CAPM which accounts for the exposure of the strategy to market risk β, but these have been less popular for use in evaluation of reinforcement learning stock price prediction models.

Chapter 3

Sentiment Analysis

Sentiment analysis for stock price prediction is a broad area with a large variety of potential approaches. The two main ways in which approaches are differentiated are the source of the textual data and the way in which that data is processed and used for prediction. For this project it will be important to use a method that can work well as an input for a reinforcement learning model alongside indicators derived from price.

3.1 Sentiment Analysis

Sentiment analysis, also known as opinion mining, involves the use of natural language processing and other techniques to extract subjective opinions from text (Hussein, 2018) . This has many applications and is an especially active area of research at the moment with the ever increasing availability of opinionated textual data available on the internet. Medhat, Hassan and Korashy (2014) provides a useful survey of recent works and general overview of the field of sentiment analysis. This survey splits sentiment analysis into two main parts, which are feature selection and sentiment classification.

Feature selection involves pulling distinct features from the text. This can be done either with a lexicon-based approach, which requires a certain amount of manual annotation, or with one of a number of statistical approaches. In this process the textual data may be converted into a Bag of Words (BOW) which can simplify the problem. However, doing this does lose the sequential context of the words so in some cases it can be better to maintain the order of the text. Maintaining order of words was shown by Li et al. (2014) to be important specifically in the area of stock price prediction from news articles with their results showing a clear improvement over the BOW approach.

The next step of sentiment classification can be done either with a machine learning approach or a lexicon-based approach. Figure 3.1 from Medhat, Hassan and Korashy (2014) gives a nice overview of the main approaches.

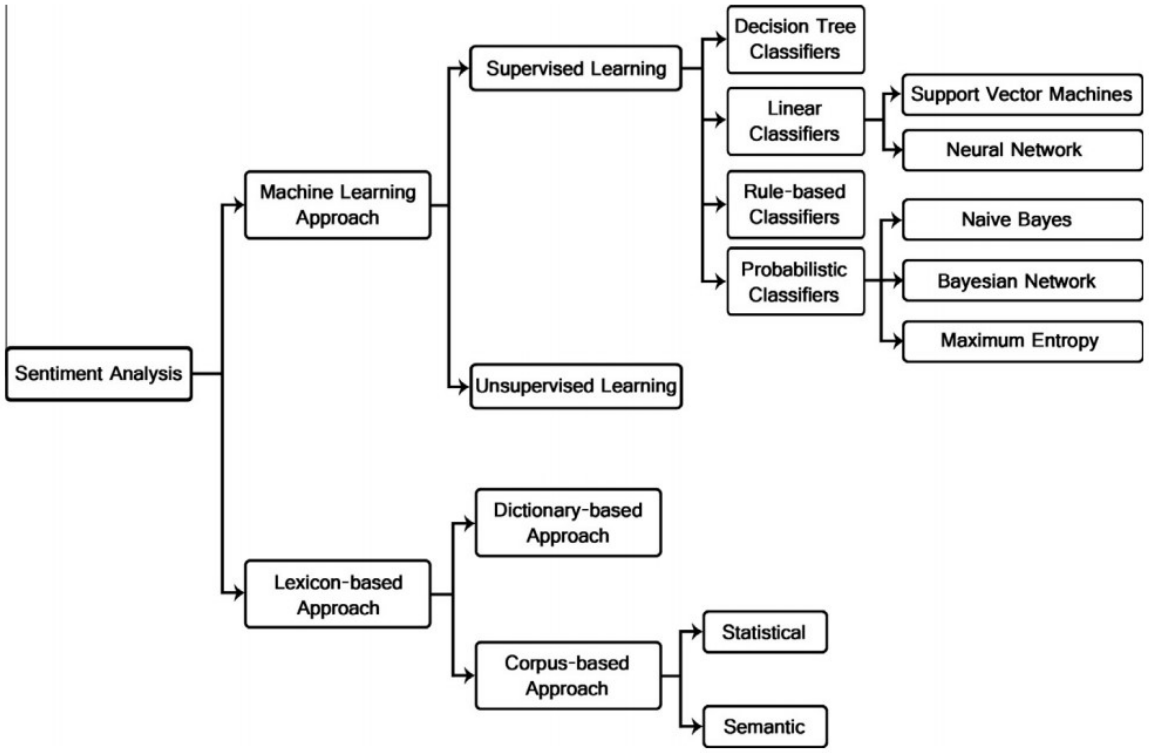


Figure 3.1: Sentiment classification techniques (Medhat, Hassan and Korashy, 2014)

3.2 Application in Stock Price Prediction

3.2.1 Information Sources

Textual data on stocks can be extracted from the internet using web scraping or from a pre prepared data archive. Generally the approach to sourcing information which can be used for sentiment analysis on stocks fall into two groups. Those that use data from social media and those that use data from news articles.

An advantage of using social media data is its abundancy on the internet. This means that sentiment can be generated on a daily basis and machine learning models can be trained on large datasets. However, a drawback is that this data can be subjective and is more likely to be inaccurate (Obthong et al., 2020). Despite this there have been successful implementations of this method. Nguyen, Shirai and Velcin (2015) used posts on the Yahoo finance message board and managed to produce an effective model. Bollen, Mao and Zeng (2011), which used Twitter posts, also produced positive results.

News articles however are more likely to be based on objective facts and more informed opinions. The disadvantage however is that the data is less frequently available but there are a number of interesting ways to overcome this problem. Nam and Seong (2019) overcomes this problem to an extent by also using news on the companies competitors. Nan, Perumal and Zaiane (2020) takes the approach of finding words that are strongly related to the stock and then including articles on these words. For instance, this paper would include articles with "Bill Gates" as news for Microsoft stock. If using a web scraping approach to obtain these articles; one approach is to use many different media websites and extract the relevant articles (Kirange

and Deshmukh, 2016; Dang and Duong, 2016). Alternatively, articles can be retrieved from sites that collate news articles that are related to given stocks. This is done by Li et al. (2014) with a financial news vendor for Hong Kong, Nan, Perumal and Zaiane (2020) with the use of the Reuters twitter account, and Koratamaddi et al. (2021) by entering search terms into Google News.

Large investment banks will often provide investment ratings for stocks. These ratings therefore have potential to be used as a valuable source of sentiment. They are also especially useful because they only consist of single words. This means that they can be converted into values simply using a dictionary approach and do not require a complex sentiment analysis model. Barber, Lehavy and Trueman (2010) shows that both these ratings and their upgrades and downgrades can used individually to produce above average returns.

3.2.2 Approaches

Shah, Isah and Zulkernine (2018) is a good example of a dictionary based approach to stock sentiment analysis. This paper generated a dictionary of words and phrases that were classified as positive, neutral, or negative. News articles could then have their sentiment scored by comparing the bag of words to the dictionary. This approach was effective in this case, but this paper was focused on the pharmaceutical industry. This is an industry where small phrases such as "FDA approval" can be very informative. It is likely that this would not be the case in a different industry.

Nam and Seong (2019) makes use of machine learning for stock sentiment analysis of news articles. The paper uses chi-square, which is a popular statistical approach to feature extraction. This representation is then classified using a Multiple Kernel Learning approach. Kirange and Deshmukh (2016) uses a dictionary to construct a feature matrix which is then used as the input for a machine learning model for sentiment classification. This paper tested Support vector machine (SVM), KNN and Naïve Bayes and found SVM to be the most effective. Dang and Duong (2016) used a combined approach for feature extraction using a dictionary and then a statistical method to reduce the dimensionality. This was then classified with an SVM.

Chapter 4

Deep Reinforcement Learning

Two of the most popular machine learning approaches that have been applied to the area of stock price prediction are supervised learning and reinforcement learning. Supervised learning involves creating a dataset of attributes and corresponding labels and then training a model on this data to predict the label given the attributes. Reinforcement learning however frames the problem in terms of states, actions and rewards instead. This review will focus on reinforcement learning but some supervised learning approaches will remain relevant as they use deep neural networks which are also used in the function approximation for deep reinforcement learning.

4.1 Reinforcement Learning

4.1.1 Approaches

Sutton and Barto (2018) is a well regarded book in the area of reinforcement learning that identifies and explains some of the key approaches. The main types of tabular reinforcement learning methods are Dynamic programming, Monte Carlo methods and Temporal difference learning. Dynamic programming methods use a full model of the environment and create a policy by repeatably iterating over the model and incrementally updating it. This uses value bootstrapping but requires a full model of the environment. Monte Carlo methods however generate policies by taking samples of the experiences in the environment but needs to complete full interactions before it can learn anything as it does not employ bootstrapping. Temporal difference (TD) learning uses both sampling and bootstrapping which makes it a very popular and versatile approach. One of the most popular algorithms that employs TD learning is Q Learning. This algorithm interacts with the environment and maintains a table of values for each combination of states and actions. Following each action that the agent takes the table will be updated using Equation 4.1. In this equation α is the learning rate, γ is the discount rate for future rewards, and $Q(S, A)$ gives the value from the table of taking action A given you are in state S.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4.1}$$

Reinforcement learning methods like this that have tables of values are known as tabular approaches. The limitation of using a table however is that as we increase the complexity of the environment the number of possible states can increase to a level where a table is no

longer a realistic approach. This is both because of memory constraints and the fact that there is no appreciation of the similarity of many states in the learning process. In order to get around this problem function approximation can be used to replace the Q table.

4.1.2 Function Approximation

Function approximation can be done with linear functions. However, in the case of deep reinforcement learning artificial neural networks (ANN) are used. The parameters of these ANNs can be updated during learning using backpropagation, which will gradually train the model. The architecture of these ANNs is an important consideration in a deep reinforcement learning problem. The design will need enough complexity so as to capture the function, but too much complexity may lead to overfitting of finite data (François-Lavet et al., 2018).

There are many different types of ANN but these are some of the most popular and significant (Mehlig, 2019). Deep Neural Networks (DNN) consist of a sequential multi layer network of perceptrons. These can be used to approximate complex functions and have a broad range of uses. A popular deep reinforcement learning algorithm that uses a DNN for function approximation is Deep Q Network (DQN), which also has a number of variations such as Double DQN and Dueling DQN. DQN uses a DNN which takes states and returns state action values. Convolutional Neural Networks (CNN) are a useful way to extract features from data and reduce dimensionality. For this reason they are very popular in applications such as image recognition. Recurrent Neural Networks (RNN) are designed in such a way that some information is maintained from one step to the next. This means they can be useful in applications where the order of data is of importance. They do however still function under relatively short memory because of vanishing gradient in backpropagation. Long Short Term Memory (LSTM) networks are an alternative that address this problem using gated cells. More recently the Gated Recurrent Unit (GRU) has also become popular as an alternative that can outperform LSTMs in terms of computational time to convergence as shown by Chung et al. (2014).

4.1.3 Policy Gradient Methods

Q Learning can be considered a value based method because its approximation function produces values for state action pairs. An alternative approach to this, called policy gradient methods, is to instead have approximation functions which given a state assigns a probability to each action of being taken. This approach can be more affective where an element of stochasticity is beneficial. For instance, in a game of rock paper scissors the best policy may be to select randomly. The drawback of this is that, similar to Monte Carlo methods, it does not utilize sampling and needs to complete full episodes before learning.

Actor critic methods are a solution to this problem that use both a policy function called the actor and a value function called the critic. A popular algorithm that implements this is Deep Deterministic Policy Gradients (DDPG). It has been shown that this algorithm can be very affective in continuous action spaces (Lillicrap et al., 2015). Figure 4.1 by Michaux (2019) illustrates the difference in function approximation between DQN and DDPG.

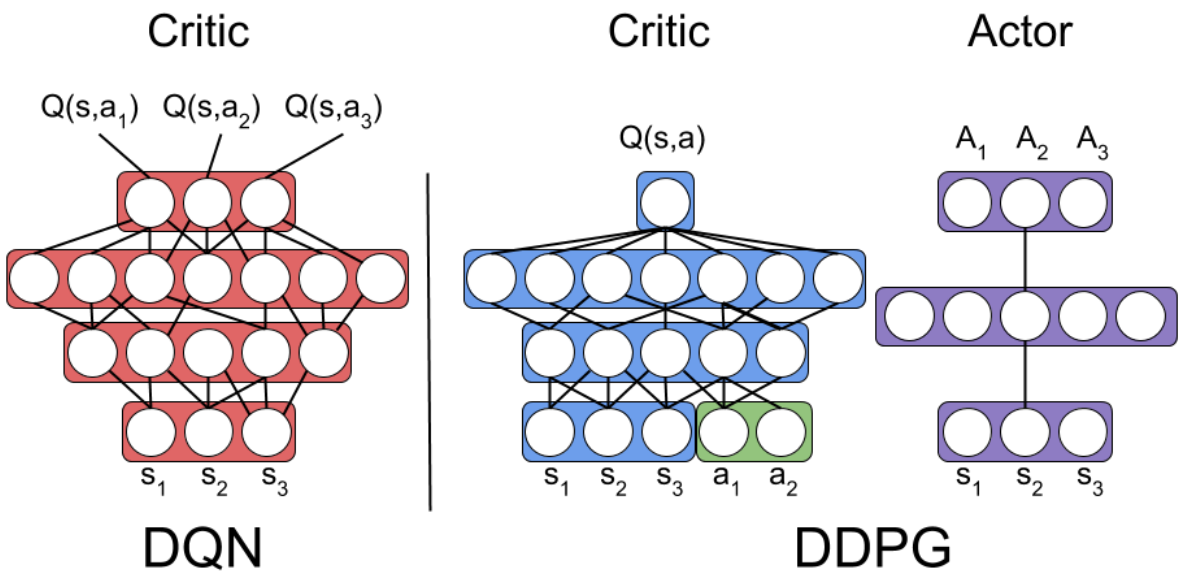


Figure 4.1: DQN and DDPG Function approximation (Michaux, 2019)

Another popular actor critic algorithm is Advantage Actor Critic (A2C) which was first presented as an asynchronous algorithm in Mnih et al. (2016). This algorithm is similar to DDPG in the use of both an actor and a critic network but their are some key differences. The main difference between the two being that for DDPG the critic is the main network, and for A2C the actor is the main network used (Huang, 2018). A2C also uses multiple workers to avoid the use of a replay buffer.

4.2 Application in Stock Price Prediction

The use of reinforcement learning in stock price prediction is a very active area of research. As such there are a number of recent surveys of the area (Hu, Zhao and Khushi, 2021; Meng and Khushi, 2019; Fischer, 2018). A far less investigated area of research has been the use of sentiment analysis as a component of the reinforcement learning model. First the application of reinforcement learning using the indicators described in Section 2.1 will be outlined and then this will be followed by the work using both reinforcement learning and sentiment analysis.

4.2.1 Indicator Based

Many different deep reinforcement learning approaches have been taken in this area. Fischer (2018) looks at the different approaches taken by number of citations and finds that critic-only such as Q learning are most popular, followed closely by actor-only and then actor-critic being the least popular. However, works such as Wu et al. (2020) did find that actor-critic methods can produce more stable returns than a critic-only method. Li et al. (2019) also produced positive results using an actor-critic method. Amongst the critic-only approaches DQN is a popular choice of algorithm. Li, Ni and Chang (2019a,b) compares this base approach against some of its variations, including Double DQN and Dueling DQN, and found that the base approach was most effective in the context of stock trading.

Another important consideration is the architecture of the neural network and with this the way in which the states and actions are framed. Due to the time series nature of stock price

data, different forms of RNNs specifically LSTMs have been popular approaches (Meng and Khushi, 2019; Hu, Zhao and Khushi, 2021). Some recent papers such as Wu et al. (2020) have also achieved positive results using GRUs instead. Another approach involved the use of CNNs. This can be done by converting data on stock prices into images. Carta et al. (2021) uses a CNN as the main function approximator in this way. Shin, Ra and Choi (2019) generates images from stock data with a number of technical indicators plotted and then uses both a CNN layer and an LSTM to make predictions.

4.2.2 Sentiment and Indicators Based

As explained in Section 3.2.2 the use of sentiment analysis performs well in stock price prediction. It would then intuitively follow that the utilization of this data in a reinforcement learning model could improve its performance. There are a limited number of papers that cover this topic and these are recently published.

Of the papers on the topic two used critic-only methods. Kaur (2017) structure their solution by having an ANN for extracting features from news articles and another ANN for trend analysis of historical price data. The output of these two networks is then formulated as a state and input into their reinforcement learning algorithm. This model however is limited by the fact that its state size increases exponentially with the size of the stock universe for the portfolio. Nan, Perumal and Zaiane (2020) is an interesting paper that uses the DQN algorithm. In this model the environment states consisted of the current portfolio conditions, price data on the stock, and a sentiment score. This sentiment score was calculated with the combined use of sentiment analysis resources as discussed in Section 5.2.2. The paper compared its reinforcement learning model with or without sentiment as an input and found that the inclusion of sentiment did improve the performance of the model. A limitation of this model is the use of a standard feedforward DNN. This meant that the model was only using the most recent price data and did not utilize information on the context of this price. Using a form of RNN would likely have produced better results.

Actor-critic approaches have also recently been used. Azhikodan, Bhat and Jadhav (2019) uses the Deep Deterministic Policy Gradient (DDPG) algorithm in their approach. For sentiment analysis a separate model Recurrent Convolutional Neural Network (RCNN) was used. This was then fed to the reinforcement learning model along with price and portfolio data. Koratamaddi et al. (2021) also uses DDPG. This model used news headlines and twitter tweets which were run through the VADER tool to produce sentiment scores. The features extracted were then formulated into a state representation alongside the closing price of the stock and input into the DDPG model.

There are some significant limitations and areas for improvement with these models. Firstly there is very limited use of technical indicators and no use of fundamental indicators. The majority of these approaches only used the closing prices of the stock. Nan, Perumal and Zaiane (2020) does use the change in price over the last 5 and last 50 days, but none of the approaches used any of the technical indicators described in Section 2.1.2. There was also no use of any fundamental indicators such as those in Section 2.1.3 which, as we have seen in 2.1.1, have been shown to improve predictive accuracy in reinforcement learning models. One other area for potential improvement to these models could be the inclusion of full news articles rather than just the news headlines. Given that these ideas have not yet been explored, it is clear that there is a need for more research in this area.

Chapter 5

Existing Resources

5.1 Data Extraction

5.1.1 Historical and Financial data

For the calculation of technical indicators and fundamental indicators historical and financial data on stocks first need to be obtained. There are a number of sources for this sort of data but by far the most popular is Yahoo Finance. Data can be extracted from this site in a number of ways. The simplest approach is simply to manually extract the data from the website. Financial data can be copied and pasted, and price and volume data can be downloaded for a given stock, frequency and date range as a csv file. A more scalable approach however is to web scrape the data using the python Requests and BeautifulSoup libraries (PSF, 2011; Richardson, 2007). This way data can be extracted easily on many stocks and updated with ease. There are also tools such as the yfinance library by Aroussi (2019) which simplify this process by providing functionality to download and present this data in an easy to use format.

An alternative approach to extracting this data could be the use of an API. A potential advantage of this would be the extraction of data which is more reliable. However, these alternatives have limitations on the number of data requests that can be made without a paid subscription. The need for credentials to use these APIs also makes any software that depends on them less easy to download and use. There are also some more expensive options that provide real time data, but this is unnecessary for purposes that are not executing real transactions.

5.1.2 Sentiment data

To extract news on given stocks a source is needed that can collate relevant articles. This is important because using an individual publisher would produce very infrequent articles on individual stocks. One potential source is Investing.com. This website provides lists of related news articles for a given stock. These articles can be extracted using webscraping. Another source of collated news articles is Google News. This approach is less specific to stock headlines as it simply returns news articles for a given search term. However by searching the companies name and providing a date range many relevant articles can be extracted.

Twitter is a frequently used source for social media stock sentiment. To extract tweets the Twitter API can be used. The standard version of this only allows for the extraction of recent

tweets, however older tweets can be extracted with either an academic or paid account. Tweets on stocks can then be found using search terms. Another source of textual data could be the comments section of the Yahoo Finance site, which could be webscraped for given stocks. Reddit have also recently become a platform for influencial stock sentiment with the unusual trading activity generated by posts on sub-reddits such as WallStreetBets.

Recommendations from financial institutions are another interesting source of sentiment on stocks. These recommendations vary in the specific terms used, but are generally to either buy, sell or hold a given stock. This form of data is easy to extract sentiment from because if its more objective nature. For instance, you could assign a value of 1 to "buy", 0 to "hold", and -1 to "sell". Yahoo Finance provides for each stock a list of recommendations that have been made about it.

Data from these sources could be extracted using web scraping with python modules such as Requests, BeautifulSoup and Selenium (SeleniumHQ, 2018). There are also some publicly available datasets. Koratamaddi et al. (2021) provides a sentiment dataset on stocks in the Dow Jones 30 from 2001 to 2019. This dataset provides values which have been derived from sentiment analysis of google news articles and twitter posts.

5.2 Data Analysis

5.2.1 Technical and Fundamental indicators

Technical and fundamental indicators can be calculated relatively easily simply by iterating through the historical stock data and calculating daily values. Tools such as the Stockstats module, by Zhuang (2020) can also be used to automate the calculation of some of these indicators and help to reduce the chance of human error.

5.2.2 Sentiment Analysis

In order to conduct sentiment analysis few papers create there models entirely from scratch and there are a number of useful resources available. Use of dictionary resources is the first main way in which this is done. Kirange and Deshmukh (2016) uses the LIWC dictionary for this which consists of 5,690 words and word stems. Li et al. (2014) uses two other dictionaries. The Harvard IV-4 sentiment dictionary is a large dictionary of 10,000 words which has many dimensions across 15 different dimension groups. The McDonald financial sentiment dictionary is a smaller and less comprehensive dictionary, but is more domain specific.

Before words can be compared against a dictionary it is often useful to first use stemming to collapse them into more distinct words and tagging elements such as whether the word was a verb or a noun and if it was preceded by a negation. Jeong, Kim and Yoon (2018) uses a popular resource for this called WordNet. There is now an updated version of this tool specifically for sentiment call SentiWordNet. Other stemmers such as the Porter stemmer and Lancaster stemmer also exist but are less appropriate for sentiment analysis and more applicable to other types of NLP.

Nan, Perumal and Zaiane (2020) uses some interesting and more complete resources including IBM Watson, TextBlob, and NLTK. IMB Watson tone analyser is an interesting tool that given some text can extract tone, at a document and sentence level, under 7 dimensions. These are anger, fear, joy, sadness, analytical, confident and tentative. Having a tool which does a

lot of the sentiment analysis work for you is very useful. However it does come at the risk of losing some of the domain specific features. In stock sentiment analysis for example this could include phrases that are rarely used outside of the domain such as "bull market". Bollen, Mao and Zeng (2011) uses some other potentially useful resources such as Google-Profile of Mood States which extracts features in 6 dimensions from text and OpinionFinder which classifies text as either positive or negative. Koratamaddi et al. (2021) uses another tool called VADER to obtain polarity scores from posts. Another resource that can be used for sentiment analysis is the Hugging Face transformers library (Wolf et al., 2019). This open source library for natural language processing offers options for sentiment analysis with varying levels of customisation. Specifically, the sentiment analysis pipeline is a simple tool which can quickly be used to label news articles as either positive or negative.

5.3 Stock Price Prediction

5.3.1 Trading Environments

Open AI Gym, first introduced in Brockman et al. (2016), provides a framework for reinforcement learning environments. Figure 5.1 shows how a reinforcement learning agent interacts with a gym environment. This standardization makes it much easier to develop and compare different reinforcement learning algorithms. Gym also provides a number of its own environments that can be used to test reinforcement learning algorithms. FinRL, presented by Liu et al. (2020), is a useful library for stock trading with deep reinforcement learning. This library provides a number of different gym compliant stock trading environments including single stock trading, multiple stock trading, and portfolio allocation.

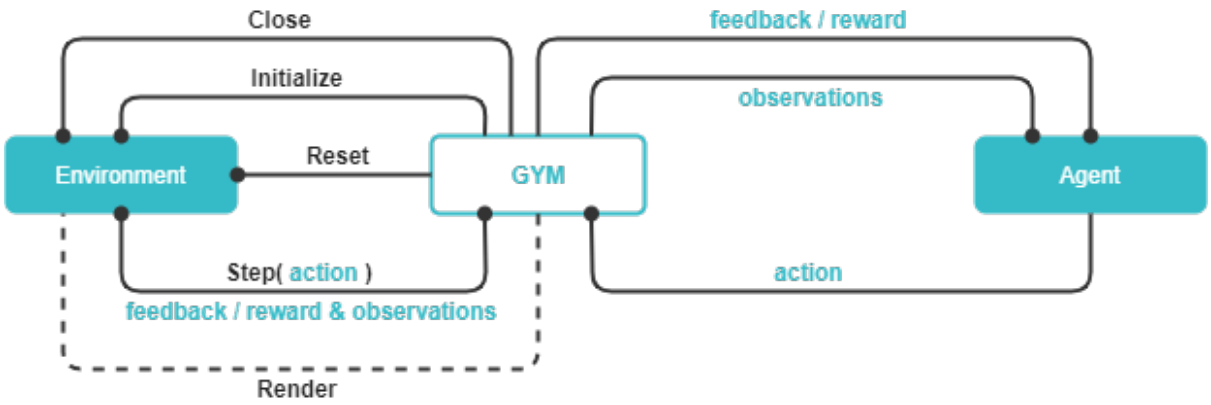


Figure 5.1: Gym environment framework (Feng, 2019)

5.3.2 Deep Reinforcement Learning Models

The most popular programming language for reinforcement learning is Python. This is due to its popularity for use with data and the availability of some useful libraries. Libraries such as Tensorflow and PyTorch can be used to create and train ANNs with custom architecture. These ANNs can then be used inside custom reinforcement learning algorithms. Alternatively a more complete resource can be used. Stable Baselines 3 provides a number of reliable implementations of deep reinforcement learning algorithms (Raffin et al., 2019). These implementations work on gym environments and as such are very easy to quickly apply to

different cases. Even with these implementations however there is still a great deal of flexibility in the specific parameters that are selected. In addition to the stock trading environments FinRL also provides a number of deep reinforcement learning implementations for use on these environments which build on those provided by Stable Baselines 3.

Training machine learning algorithms can be a long and computationally expensive task. If available, using a GPU instead of a CPU has the potential to significantly speed up this process. Google Colab does provide free access to GPU use. There are however significant limitations on the duration of their use, so for longer training periods an alternative will be required.

5.3.3 Backtesting and Evaluation

In order to evaluate the performance of trading strategies it is useful to produce visualisations of the data. Matplotlib is a popular python library for visualisations and can be used to plot the performance of different trading strategies. Pyfolio by Quantopian (2015) is a python library that can be used to evaluate the performance of trading backtests. Given the returns produced by a trading strategy Pyfolio can be used to calculate many useful evaluation metrics such as the Sharpe ratio and the annualised return. The library also provides many specialised visualisations of the performance such as returns and volatility comparisons against benchmarks.

Chapter 6

Methodology

As covered in Section 1.2, following the completion of the literature and technology review the main objectives of this project are as follows.

- Develop software for construction of portfolio allocation environments that use both technical indicators and sentiment features.
- Compare the performance of agents using these features individually and together when used in a portfolio allocation environment.
- Compare the performance of critic-only and actor-critic agents in a portfolio allocation environment.

This section will outline the approach towards the software development and the subsequent experimentation.

6.1 Environment and Evaluation

In this project the environment will be modelled as portfolio management. At the beginning of each episode the agent will have a set amount of money in the portfolio. Episodes will consist of one full run of the training data and each day the agent will decide on the portfolio allocation by providing the weight assigned to each stock. At each state the agent will receive only information that was made available either before or on the current day. The agent will receive rewards based on the absolute change in the portfolio value from the time of the trade to the following day.

The performance of agents will be measured and compared against other agents using the total return achieved and the Sharpe ratio for the reasons explained in Section 2.2.2. As a benchmark we will use the performance of an untrained agent.

6.2 Technical and Fundamental Indicators

During the process of producing the initial actor-critic deep reinforcement learning model there will be a certain amount of testing to identify which indicators are most effective as inputs. It may also be necessary to limit the number of indicators so as to restrict the dimensionality of the problem and improve learning speed. Of the technical indicators from Table 2.1 the ones of most interest are MACD, RSI, and Bollinger Bands. The price and financial data required

to calculate these indicators will be retrieved primarily from Yahoo Finance as described in Section 5.1.

The decision has been made to exclude fundamental indicators for some key reasons. Firstly, the precise date that financial data was actually released is difficult to extract with web scraping. Without this information we cannot know when in the time series to update this data; and would therefore risk providing the agent with information before its publicly availability. Secondly, the availability of this data from free providers such as Yahoo finance is limited to the last 4 years. This would significantly reduce the available dataset.

6.3 Sentiment Analysis

For sentiment analysis the approach taken will be similar to that taken by Nan, Perumal and Zaiane (2020); Koratamaddi et al. (2021). This will involve using tools that extract sentiment directly from text as described in Section 5.2.2. Depending on the success of this approach adjustments or other more involved approaches will also be considered. For extraction of sentiment from ratings, a dictionary of values will be constructed from scratch as the number of possible words is fairly limited.

To retrieve relevant news articles two main sources will be considered. Google news can be used by simply entering a stock name and scraping the resultant news articles along with their respective dates posted. These can then be sorted and used. Investing.com is another alternative that have a news section that is populated by the stock you are currently looking at. This data can be extracted in a similar way but may perhaps be more relevant. Both these sources will be considered in the implementation stage of the project.

6.4 Deep Reinforcement Learning

The primary deep reinforcement learning algorithm used for this project will be either DDPG or A2C, depending on performance, because of their ability to handle continuous action spaces. DQN will also be used as a critic-only approach for comparison. Function approximation will be done using either a GRU or a simple DNN. State representation used by the model will consist of the technical indicator, and the daily sentiment values. Each stock in the stock universe will receive an action in the form of a value. The magnitude of the action will represent the preference for this stock. The reward from an action will be equal to the absolute difference between the value of the portfolio at the time of the action and the value of the portfolio the following day. The parameters used in this model will be based on those used in previous work and an initial parameter search experiment.

6.5 Software

6.5.1 Requirements

The software for this project will need to be able to extract, process and store data. It will also need to include both a reinforcement learning environment and deep reinforcement learning agents to train on this environment. The full requirements for this software are listed below.

1. Data Extraction

- Extraction of historical price and volumes data on given stocks.
- Extraction of historical headlines on given stocks.
- Extraction of historical investment ratings on given stocks.
- Should function for any given stock, not just those used in experiments.

2. Data Processing
   - Calculation of technical indicators from price data.
   - Calculation of daily sentiment values from news headlines.
   - Calculation of daily sentiment values from investment ratings.
3. Data Storage
   - Must be able to store stock data.
   - Must be able to retrieve and reuse this data.
4. Stock Portfolio Allocation Environment
   - Must follow Gym framework.
   - Must work for both continuous and discrete actions spaces.
   - Must track the performance of episodes.
5. Deep Reinforcement Learning Agents
   - A2C agent that works with portfolio allocation environment.
   - DQN agent that works with portfolio allocation environment.

6.5.2 Design

The UML diagram in Figure 6.1 gives the planned design of the software. Agents will be trained on portfolio allocation environments which in turn will contain a number of `Stock` objects.

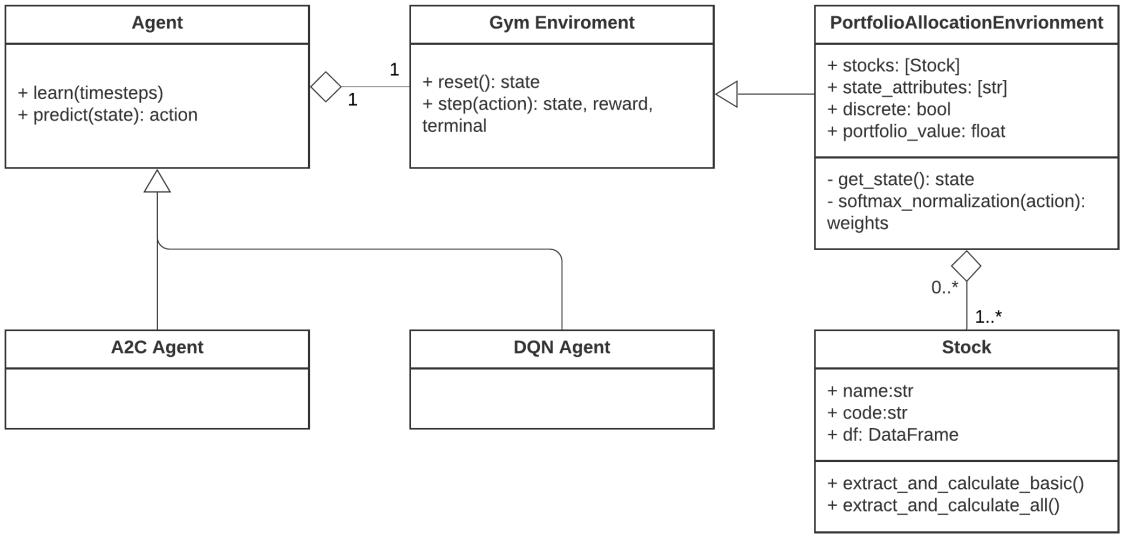


Figure 6.1: UML Diagram of Design

The process for using this software will first involve creating `Stock` objects. For each of these stocks the `extract_and_calculate_all` method can then be used to extract data, calculate indicators, and then store this data in the `df` attribute. Once extraction is complete the stocks can be stored in memory. Next a `PortfolioAllocationEnvironment` object can be

instantiated with these stocks in the constructor. With the portfolio objects the agents can then be trained. Finally, a new environment can be created with different data and used to test the trained agents.

The implementation of this project will be done using Python and will require a number of dependencies and resources. For extraction of data the Bs4, Requests, and Selenium libraries will be used to web scrape data. The extraction of sentiment from news headlines will require sentiment libraries. For the deep reinforcement learning agents either TensorFlow or PyTorch will be used to construct the function approximation ANNs. For training of the agents a GPU server will be used, which will significantly accelerate training.

6.6 Experimentation

The experimentation in this project will be based on the following three hypotheses:

Hypothesis 1 (H1) *Deep reinforcement learning models can produce higher overall return than untrained agents for a given universe of stocks using either technical indicators or sentiment analysis individually.*

Hypothesis 2 (H2) *A model using both inputs will produce higher overall return than the models using only one of the inputs.*

Hypothesis 3 (H3) *An actor-critic deep reinforcement learning approach will produce higher overall return than a critic-only approach.*

To test these hypotheses the following 4 models will be implemented which will all be trained and tested under the same conditions.

1. Actor-critic agent with technical indicators.
2. Actor-critic agent with sentiment analysis scores.
3. Actor-critic agent with technical indicators and sentiment analysis scores.
4. Critic-only agent with technical indicators and sentiment analysis scores.

The following comparisons will then be made in terms of both total return and Sharpe ratio achieved to test the hypotheses.

1. Agents 1 and 2 will all be compared against untrained agents to test H1.
2. Agent 3 will be compared against agents 1 and 2 for H2.
3. Agent 3 will be compared to agent 4 to test H3.

6.7 Limitations

One limitation with using a portfolio allocation environment is that it does not account for transaction costs. This means that the agents will have an advantage over a real life trading strategy. Despite this however a successful portfolio allocation model could still be used as a component of a live trading strategy. Another limitation that arises from the introduction of sentiment data is the reduction in training and testing data. If only technical indicators were used we would be able to use intraday price data, however due to the more limited frequency of sentiment data we are limited to a daily data approach.

Chapter 7

Implementation

7.1 Stock data

The first stage of implementation was to design and build a system to download and prepare data for given stocks and time periods. In order to do this a class was created called Stock as seen in Appendix B.2. The constructor of this class takes a start date, end date, and details on a given stock such as its name and stock code.

7.1.1 Data Extraction

When new stock class objects are instantiated the price data is extracted from the Yahoo Finance website using the Requests and Beautiful Soup libraries and a data frame is constructed to store this and other daily data on the stock. Using the `extract_investment_ranking_data` method recommendations are also extracted from the Yahoo finance website in a similar way. This uses a regular expression provided by Button (2019) to extract the json data which contains the recommendations from the page.

The `extract_news_data` method consists of two different potential sources which can be selected with boolean parameters. The first source is Investing.com. News article headlines are extracted from here with the same libraries by iterating through pages of the news section for each stock. The other source used is Google News. For the extraction of these articles the Selenium module and Chrome webdriver are also used. This approach is more effective because of the dynamic nature of the website. In order to get an even distribution of articles the method extracts a given number of articles for each year of the full date range as shown in Figure 7.1. By default this is set to 100 articles. In order to avoid captcha forms a 15 minute delay is needed between each stock when extracting this data. Extraction of the content of articles was also considered. For the Investing.com articles this was possible for internal articles, however it is very difficult to accurately extract content from external links because the web pages were all in different formats. The same problem of course applied to all Google News articles as these results were exclusively external links. For this reason article content is not extracted by the class; only the news headlines.

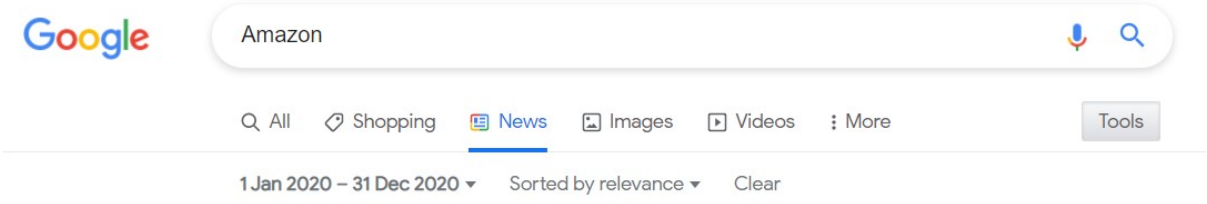


Figure 7.1: News Search Example

7.1.2 Data Processing

Although tools such as Stockstats can help to automate the calculation of technical indicators; in this project it has been implemented manually for improved flexibility. The `calculate_technical_indicators` method for the stock class iterates through the daily data to calculate a number of technical indicators for each time step.

Five different technical indicators were selected for use in this project. These were MACD, Signal Line, Normalized RSI, Standard Deviations from Mean and Relative Volume. The Moving Average Convergence Divergence (MACD) as mentioned in Section 2.1.2 is the difference between the short and long EMA of a stock and the Signal Line is simply the exponential moving average of the MACD. The reason that both of these indicators were selected was because in trading a technical signal is triggered when these two lines cross. As such these two values being close together could be identified by the agent as a signal. The RSI was also included in the selection only its value was normalized to optimize for use in deep learning. Partially inspired by Bollinger Bands the Standard Deviations from mean was also included as a more compact way to provide this information to a deep learning model. The final technical indicator was the Relative Volume. The rationale for this was that it could be used by the model in combination with other features to confirm the significance signals (Rockefeller, 2019). For instance if there is a high trading volume following a sentiment signal then perhaps this sentiment is more likely to influence the stocks price.

A number of different sentiment analysis resources were identified in Section 5.2.2. During implementation the output of several of these resources were compared. Table 7.1 provides a comparison of outputs produced by the resources with an input of "Amazon posts biggest profit ever at height of pandemic in U.S.". The HuggingFace sentiment analysis pipeline uses neural networks and as such takes a little longer to calculate sentiment than some of the other resources. However, it did appear to produce relatively accurate sentiment scores. TextBlob was one of the more simple approaches which simply looks for words that it has sentiment for and then returns the average for the text. This is very quick but did not produce very good results, with the headlines often not having enough words to even get a score. Vader is a rules based sentiment model designed primarily for social media text (Hutto and Gilbert, 2014). It is a fast method and performs reasonably well at classifying news headlines. The last resource used was IBM's tone analyzer. This tool returns a list of identified tones and a value associated with them; making it less easy to use in our model than the other resources. It is also accessed using an API which has a usage limit for the free version. As such this resource was excluded.

Name	Response
HuggingFace	{'label': 'POSITIVE', 'score': 0.9848056435585022}
TextBlob	{'polarity': 0.0, 'subjectivity': 0.0}
Vader	{'neg': 0.0, 'neu': 0.662, 'pos': 0.338, 'compound': 0.5574}
IBM	{'tones': [{'score': 0.543112, 'tone_id': 'confident', 'tone_name': 'Confident'}]}

Table 7.1: Sentiment Analysis Comparison for "Amazon posts biggest profit ever at height of pandemic in U.S."

The processing of news headline data presented a number of challenges. One issue that arose with the headlines extracted from Investing.com was that for the very well known stocks many of the articles listed had little to do with the stock itself and were only included because the stock was mentioned at some point during the content of the article. Due to this and the increased variation in availability of articles Google news was used instead. Another issue with headlines from both sources, and a more general challenge of sentiment analysis, was that positive sentiment in a headline about a stock does not always mean that sentiment is directed towards said stock. For instance the following headline was scraped from a Google News search for Walmart "Facebook is now worth more than Walmart". This headline returns a high confidence label of "POSITIVE" using the Hugging Face classifier even though the positive sentiment in this headline is of course actually directed towards Facebook.

In order to derive sentiment from investment ratings we need a way to convert the rating words into values. As mentioned in Section 3.2.1 this can be done with the use of a dictionary of words and their respective values. Due to the way that raw ratings data is presented, for change in rating this is very simple because there are only 5 possible words used to represent the change. We can therefore just set "Initiate", "Maintain" and "Reiterate" to a value of 0, "Up" to a value of 1, and "Down" to a value of -1. The actual ratings however had far more possible words. The reason for this being that different financial institutions use different words for their ratings. For instance, a rating of "Outperform" and "Buy" might be two different institutions way of saying essentially the same thing as seen in Figure 7.2. To construct this dictionary a set of all ratings that had been used for the top 100 stocks in the S&P 500 was constructed. These ratings were then all manually labelled as either -1, 0 or 1 as seen in Appendix B.4.

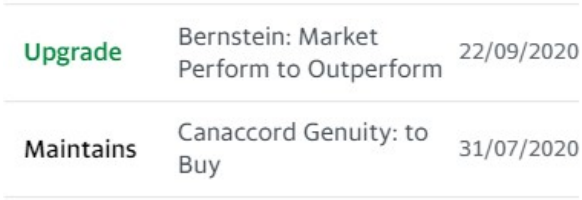


Figure 7.2: Ratings Example

Individual headlines and recommendations will occur less frequently than daily price data on a given stock. For this reason if we just recorded sentiment values on the days in which they occurred many days would not have any sentiment value. This approach would also be ineffective because it would mean that an agent would neglect all sentiment that did not

happen on the same day. As a result a portfolio allocation agent might increase the weight of a stock on the day that it received positive sentiment for that stock but the following day it would completely neglect this information even though it is most likely still of importance. In order to address this issue we need a way to translate scattered sentiment values into daily sentiment scores which incorporate all recent sentiment on the stock. To do this a sentiment decay approach was used. This approach was defined by Equations 7.1 and 7.2 where S is either the sum of sentiment values for a given day, or 0 if there is no sentiment available, and λ is a decay constant between 0 and 1.

$$V_0 = 0 \tag{7.1}$$

$$V_{n+1} = \lambda V_n + S_n \tag{7.2}$$

λ is an important parameter as it determines how long the agent will continue to account for a given point of sentiment. If this constant is too high then the importance of more recent sentiment will be lost, however if it is too low we will still have the issue of short memory. For the experiments in this paper λ was set to a value of 0.9. This value was selected as the significance of sentiment remained high for the following week of trading days at $0.9^5 = 0.59$, but by the end of a month of trading days was relatively insignificant at $0.9^{20} = 0.12$.

An issue that arose for sentiment from both headlines and recommendations was the high variation of frequency amongst the stocks. For instance, on the time of writing Apple returns 804 investment ratings whereas Exxon Mobil returned only 178. This is less of an issue when sentiment on average is around 0 however this is often not the case. This means that certain stocks might be favoured just for receiving sentiment more often, even if this sentiment isn't actually any better than that of other stocks. There are two main ways in which this problem was addressed in the implementation. The first was to extract a uniform number of headlines from google news. This meant that all stocks would have the same frequency of sentiment. The other approach was to find ways to make the average sentiment score close to zero to negate the advantage of sentiment frequency. With the investment ratings it appeared that when values were manually assigned to recommendations, for instance "Buy": 1, "Hold": 0, and "Sell": -1, that the average score was fairly high at 0.61. However if instead we use the change values, such as "Upgrade": 1, "Maintain": 0, and "Downgrade": -1, we get an average much closer to 0 at 0.01.

7.1.3 Data Storage

As seen in Appendix B.3 functions were written to store data within stock objects. The python dill library is used, which allows for the storage of python objects. This approach rather than saving the data frames separately allows for increased flexibility. The `retrieve_stocks_from_folder` function can extract all stock objects stored in a folder and can be used to load the experimentation dataset.

7.2 Environment

The next stage of implementation was to build environments for the agents to interact with. As mentioned in Section 5.3.1 there are a number of ways to design stock trading environments.

In this paper a portfolio allocation approach is used. The class for this environment called `PortfolioAllocationEnvironment` is available in Appendix B.1 and inherits from the gym `Env` class. This class was partially inspired by the portfolio allocation environment used by Liu et al. (2020) and uses a similar structure for continuous actions.

To instantiate a `PortfolioAllocationEnvironment` object the constructor takes the three parameters `stocks`, `state_attributes` and `discrete`. The `stocks` parameter takes a list of stock data frames. These data frames should all be over the same time period and can be taken from the `df` attribute of `Stock` objects. The `state_attributes` takes a list of column names that will be used for the state representation of the environment. The value of this parameter is that agents can be easily trained and compared using different feature combinations. The final parameter `discrete` is simply a boolean value to determine whether actions are discrete or continuous. For DQN agents this is set to true and for A2C agents this is set to false. When one of these environments is reset the portfolio value is set to \$1,000,000 and the date index is set to 0. There are also a number of attributes that are used to store a history of returns, portfolio values, and actions which can later be used to measure performance.

The `step` method is used by the agent to interact with the environment. This method takes an `action` as its parameter. If not discrete this action is in the form of an array of preferences for stocks. These preferences (p) are then converted into weights (w) using Softmax normalization to ensure the weights sum to a total of 1, as seen in Equation 7.3.

$$w_i = \frac{e^{p_i}}{\sum_{j=1} e^{p_j}} \tag{7.3}$$

Due to the fact that fresh weights are provided in the action no information on previous weights is required in this method. Using Equations 7.4 and 7.5 the portfolio value (P) is then updated using the previous and current close price (c) values. The reward (R) can then also be calculated using Equation 7.6 which simply returns the change in portfolio value.

$$r_i = \sum_{j=1} w_{j_i}(\frac{c_{j_i}}{c_{j_{i-1}}} - 1) \tag{7.4}$$

$$P_i = (r_i + 1)P_{i-1} \tag{7.5}$$

$$R = P_i - P_{i-1} = r_i P_{i-1} \tag{7.6}$$

If however the environment is discrete then the action is simply a number to represent which stock will be held for the next time step. The return for this action is given by Equation 7.7 as only stock j is being held.

$$r_i = \frac{c_{j_i}}{c_{j_{i-1}}} - 1 \tag{7.7}$$

The state representation is obtained using the `get_state` method which returns an array of arrays that consists of an array for each feature that contains values for each stock. Finally the method checks to see if the end of the data has been reached and then returns the state, the reward, and whether a terminal state has been reached.

7.3 Trading Agents

The agents used in this project were constructed using models from the stable baselines 3 library. The reason for this was to provide a fairer comparison between the models. The algorithms used were A2C for the actor-critic model and DQN for the critic-only model. For function approximation a DNN was used. The reason for using a DNN over an RNN was that the features in the environment were all calculated in part using data from previous days. This meant that the context of the new price, volume and sentiment was already accounted for. The parameters used for the A2C model were initially set to the default values and then refined with a parameter search, as seen in Section 8.2. For the DQN model most parameters were set to default values, such as the learning rate which was set to 0.0001, as these appeared to perform well. The gamma parameter however was set to 0. This was because of the nature of the environment as explained in Section 8.2. Due to the limited dataset the buffer size was also reduced to a maximum of 100,000.

7.4 Performance Evaluation

To track performance the `PortfolioAllocationEnvironment` class calculated a number of metrics. During each episode in the environment a full record of the returns is kept. At the end of the episode the final value, annualised return and Sharpe ratio are then all stored in class attributes. For the Sharpe ratio the risk free rate of return was set to 2% as this was the approximate average of the 10 year US Treasury rate over the experimental data period (MacroTrends, 2021). During the experiments final values are saved to excel files for all of the episodes. For the testing run the annualized return and Sharpe ratio are also stored. With these excel files the `plots_and_stats` function seen in Appendix B.5 can then be used to calculate further statistics, such as error bars and significance testing, and produce graph images.

Chapter 8

Experimentation

8.1 Environment Details

With the flexibility of the `PortfolioAllocationEnvironment` class some design decisions needed to be made about the way in which the environment was constructed for the experimentation.

Firstly, a decision had to be made on the specific stocks that would be used. The main consideration for this decision was the availability of data. For the price and volume data this was not a significant concern as it was easy to extract on most stocks. For collection of sentiment data however it was more important that the stocks were more well known. Specifically, for less well known stocks the quality of news headlines extracted would be much lower and more likely to be unrelated to the stocks themselves. Investment recommendations were also much less frequent in less well known stocks. The decision was initially made to use the top 100 stocks by market capitalization in the S&P 500. As listed at FKnol (2021). Following the extraction of news headlines on these stocks however this was reduced to the top 50 for the experiments which incorporated sentiment in order to preserve data quality.

Next an appropriate date range had to be selected for the data. For each stock in the top 50 S&P 500 the earliest available data for price and investment ratings were recorded to identify an appropriate start date. The latest starting price data for a stock was ABBV with a start of 2013-01-02. For the investment recommendations the latest start was from TMUS and was 2013-03-11. For the news headlines there was of course no fixed start to the data. However, the quality of the headlines extracted did reduce for earlier periods. Despite this, news headlines from 2014 onwards for the top 50 S&P 500 did appear to be relevant and of high enough quality to use. Given this availability of data the selected full data range for the experiments was 2014-01-01 to 2021-01-01.

Finally, the data had to be divided so as to produce separate environments for training and testing of the agents. Due to the way that some of the technical indicators are calculated they require a certain amount of data from previous days. The daily sentiment values are also better after some time has passed because they are also partially calculated with previous daily data. To deal with this issue the training environment started 100 days after the start of the data. The selected ranges were 2014-05-28 to 2017-12-20 for the training period and 2017-12-20 to 2020-12-31 for the testing period.

8.2 Experiment 1: Parameter Search

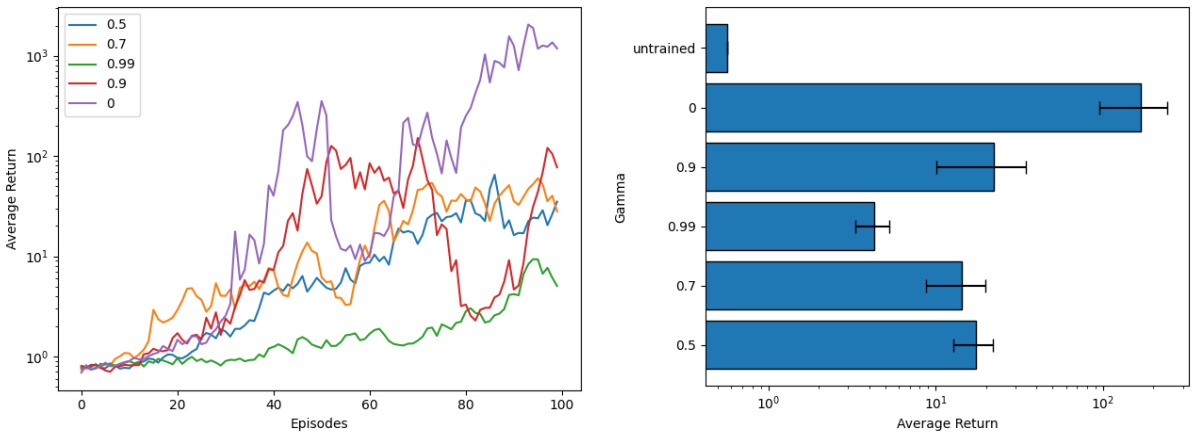
During the initial informal experimentation it was difficult to identify the most effective parameters to use for the deep reinforcement learning agents. Specifically, the best values for the learning rate and gamma parameters. This was partly due to the long training time but also to a significant extent the uncertainty surrounding the usefulness of the individual state features. To address the issue of feature usefulness a new feature was introduced that is objectively useful for reward maximisation. This feature was simply the return that the agent would get from each stock in the stock universe over the next time step and was called "cheat". Therefore all the agent would need to learn for an effective strategy would be to output high action values to stocks with high feature values.

8.2.1 Approach

The approach to this experiment was to use a stable baselines A2C deep reinforcement learning agent, with the portfolio allocation environment using the cheat attribute for state representation. Two sub experiments were then conducted using learning rate and gamma as independent variables. For each sub experiment 5 values were selected for the independent variables and for each of these values 10 agents were trained for 100,000 timesteps in the training environment and then tested in the testing environment. For the first experiment gamma was fixed at a value of 0 and the learning rate was set to 0.0001, 0.0005, 0.001, 0.005 and 0.01. For the second experiment learning rate was fixed at 0.001 and gamma was set to 0, 0.5, 0.7, 0.9 and 0.99. The fixed parameters in these experiments were selected because they appeared to perform well in preliminary tests. Following the results for the learning rate sub experiment an additional sub experiment was conducted to further narrow down the value of this parameter with the rate set to 0.0003, 0.0004, 0.0005, 0.0006 and 0.0007. For all experiments the performance of an untrained agent was also used for comparison in the testing environment.

8.2.2 Results and Discussion

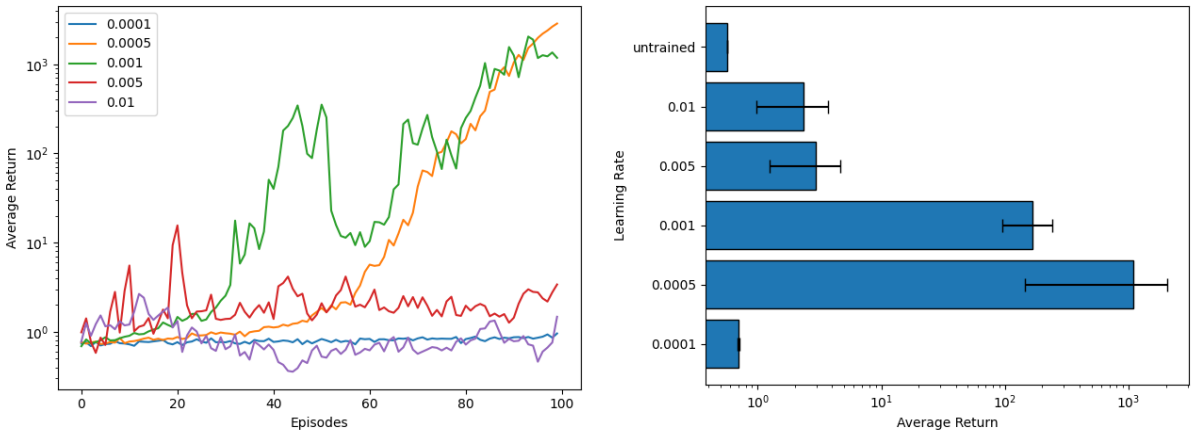
Figure 8.1 shows the results of the gamma value search. The agent with gamma set to 0 appears to have the best performance both throughout training and in the testing environment. This is likely due to the way in which the environment is designed. Specifically, the fact that the agent can change weightings instantly without any consequence means that old actions do not have a significant effect on later rewards but they do have a very significant effect on the immediate reward. Due to this fact gamma will be set to 0 for the remaining experiments.



(a) Training (b) Testing

Figure 8.1: Gamma Broad Search

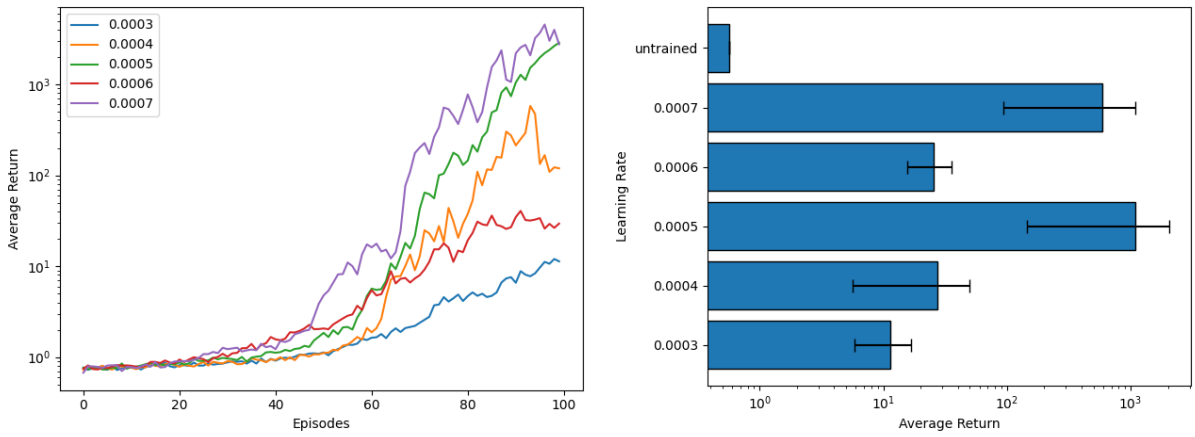
Figure 8.2 shows the results of the initial learning rate search. The two best performing values here were 0.001 and 0.0005. The value of 0.0005 performed the best in training and testing. However, performance of 0.001 was within the error bars of the average final value achieved by 0.0005. Despite this the value of 0.001 does appear to be very volatile during training compared to 0.0005.



(a) Training (b) Testing

Figure 8.2: Learning Rate Broad Search

Figure 8.3 shows the results of the final parameter search sub experiment to further narrow down the optimal learning rate value. The value of 0.0005 still appears to perform the best in testing. The value of 0.0007 however does marginally outperform 0.0005 during training. Given the testing results, and the fact that increased learning rate appears to increase the volatility of returns, 0.0005 will be used as the learning rate for A2C agents in the remaining experiments.



(a) Training (b) Testing

Figure 8.3: Learning Rate Narrow Search

8.3 Experiment 2: Technical Indicator Effectiveness

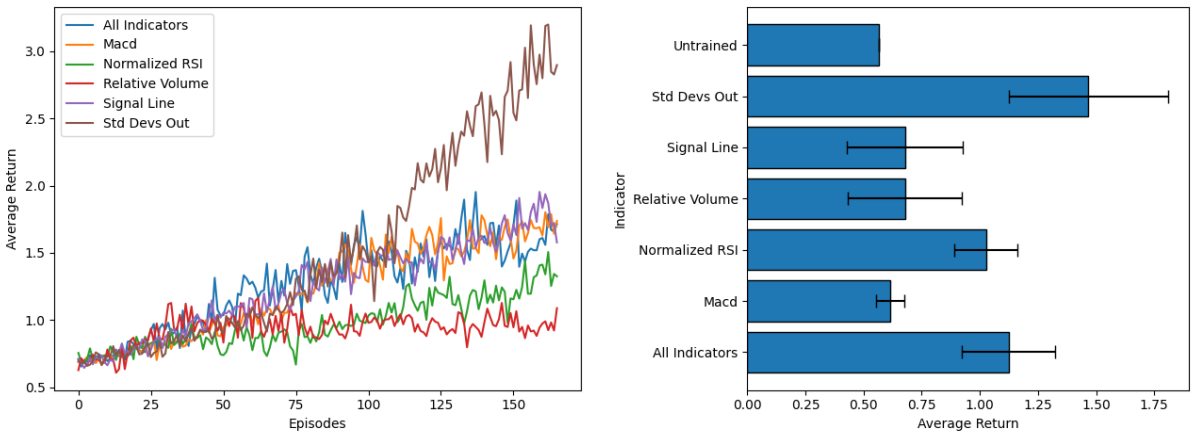
To test Hypothesis 1, and the effectiveness of technical indicators as features, all indicators were compared against each other individually and in combination as well as an untrained agent.

8.3.1 Approach

For this experiment the stable baselines A2C deep reinforcement learning agent was used again with 0.0005 for the learning rate and 0 for gamma, as identified in Section 8.2. For each individual technical indicator and a combination of all technical indicators 10 agents were trained for 150,000 timesteps and then tested in the testing environment.

8.3.2 Results and Discussion

The results of the comparison in Figure 8.4 show that some of the technical indicators are outperforming the untrained agent. The error bars here represent one standard deviation of the sample means calculated using the sample size and standard deviation. Specifically, the standard deviations out indicator produced relatively high returns during both training and testing. Normalized RSI also produces a high average final value in the testing. The other individual technical indicators do produce higher average returns than the untrained agent however the untrained agent does remain inside the error bars.



(a) Training (b) Testing

Figure 8.4: Technical Indicator Comparison

To test Hypothesis 1 we form a null hypothesis (H_0) and an alternate hypothesis (H_A). As the standard deviation is negligible for the untrained agent we will use the sample mean as an approximation of the population mean. We then use the fact that this mean return is 0.57 to form the following hypotheses. $H_0 : \mu \leq 0.57$ and $H_A : \mu > 0.57$.

$$Z = \frac{\overline{x} - \mu}{\sigma} \tag{8.1}$$

We can then use the sample mean ($\overline{x}$), the sample standard deviation (σ) and Equation 8.1 to calculate an approximate Z value of 2.75. With this we get an approximate p value of 0.0035. As $0.0035 < 0.05$ we can reject the null hypothesis.

These results therefore provide evidence to support Hypothesis 1. It is also likely that higher returns could be achieved with additional training time. Especially considering the fact that the training time required to achieve best performance will be higher due to the increased dimensionality of combining the features.

8.4 Experiment 3: Sentiment Features Effectiveness

8.4.1 Approach

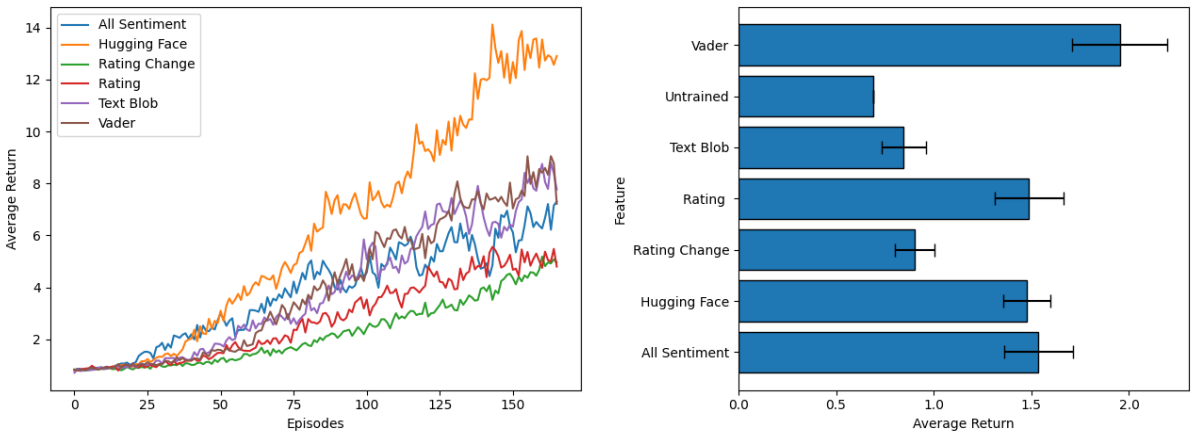
For this experiment the same agent and parameters were used. Instead of all 100 stocks however the stock universe was limited to the top 50 stocks by market capitalization. As explained in Section 8.1 This decision was made because the quality of headlines extracted appeared to drop off in the later stocks because of the reduced coverage from the media. Again 10 agents were trained and tested for each individual feature and a combination of all features.

8.4.2 Results and Discussion

The results in Figure 8.5 show that all features appear to have outperformed the untrained agent in testing. Of the features derived from sentiment analysis of news headlines the

most effective appear to be Vader and Hugging Face with TextBlob producing only marginal outperformance. This is likely due to the high simplicity of the TextBlob model. Hugging Face performed well as expected but did produce an average return less than Vader. The high performance of the feature using Vader was an interesting result and may be explained in part by the fact that this model was designed specifically for short form text such as social media posts. This model also had the advantage of producing continuous values for sentiment rather than just positive and negative labels like those used by Hugging Face.

Both recommendation based features outperformed the untrained agent with the Rating agent performing the best. This is interesting because Rating Change could be considered to be more telling of recent sentiment whereas actual Rating is more indicative of long term expectations for the stock. Rating Change however does appear to still have a high rate of learning at the end of the training period whereas Rating appears to have plateaued.



(a) Training (b) Testing

Figure 8.5: Sentiment Features Comparison

To test Hypothesis 1 we again form a null hypothesis (H_0) and an alternate hypothesis (H_A). Given the mean return for the untrained agent is 0.69. $H_0 : \mu \leq 0.69$ and $H_A : \mu > 0.69$. With our results for the All Sentiment agents a Z value of 4.8 is calculated. Giving us a corresponding p value of less than 0.0001. We can therefore reject the null hypothesis.

With Experiments 2 and 3 we have provided evidence to support Hypothesis 1 that an agent can outperform an untrained agent with either sentiment features or technical indicators individually.

8.5 Experiment 4: Combined Features Effectiveness

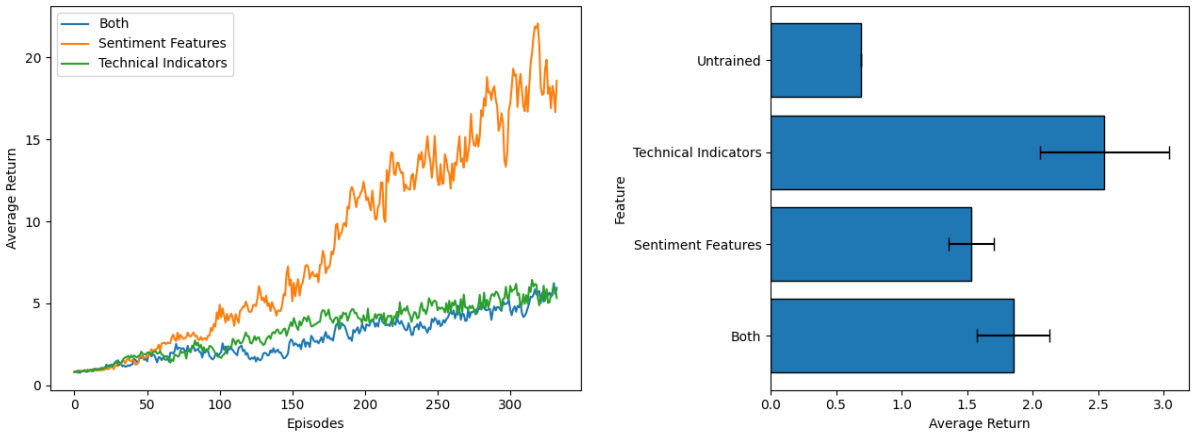
8.5.1 Approach

In this experiment we hope to test Hypothesis 2. The same model, parameters and stocks were used but the training period was increased to 300,000 timesteps. The reason for this was to allow these agents sufficient time to learn given there increased dimensionality. Following the results of this experiment a further sub experiment was conducted with a more refined selection of features. Only the three most effective technical indicators and three most effective

sentiment features were used. The training period was increased to 500,000 timesteps and the number of repeats was decreased to 5 to facilitate this longer training period.

8.5.2 Results and Discussion

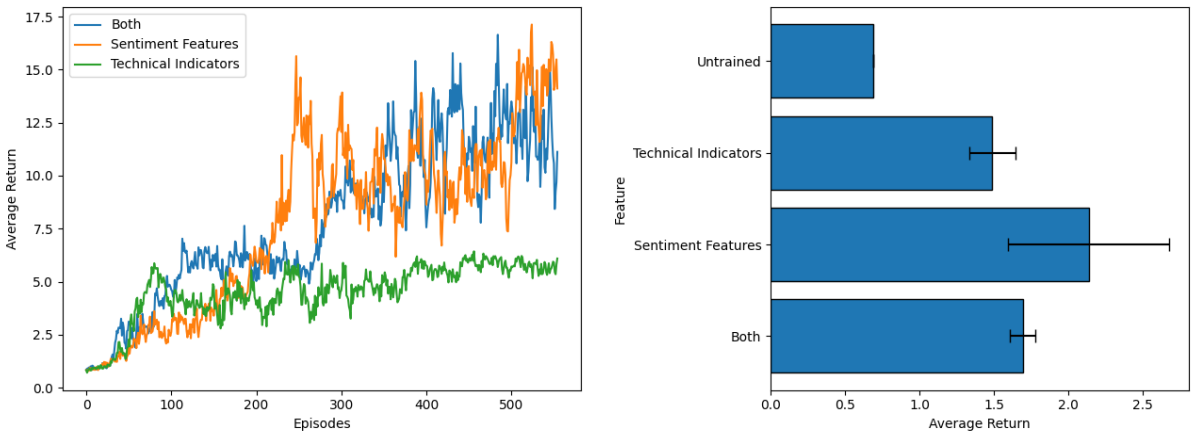
The results of this experiment in Figure 8.6 show that the agents with all features combined did not outperformed the agents with technical indicators alone. Interestingly there was a significant improvement in the performance of the technical indicator agents with the additional training time. Which we can see when we compare the agents to those with the same features in Experiment 2, shown in Figure 8.4. This may help to explain the fact that the agent with all features did not perform best because it may well have benefitted from additional training time. In the Training data we can also see a higher learning gradient in this agent at the end of the training period. Another observation from the training performance is that the sentiment features produced very high return relative to the other agents. This is interesting because in the testing it performed the least well. This may be a sign that the model has started to overfit the data in the training period.



(a) Training (b) Testing

Figure 8.6: Combined Features Comparison

Results from the refined features comparison in Figure 8.7 find that this time the Sentiment Features performs best. The differences in these results however are not statistically significant. In both sub experiments the agent with both types of feature has produced performance somewhere between the technical and sentiment feature agents. This may suggest that the model is not learning any ways in which the two feature sets complement each other.



(a) Training (b) Testing

Figure 8.7: Combined Features Refined Comparison

As an additional metric for comparison in this experiment the average Sharpe Ratio for the groups of agents was also calculated and plotted in Figure 8.8. It would appear that the difference between the performance of the agents and the untrained agent is much less significant with this metric. It must therefore be the case that the returns of the trained agents were on average more volatile than those of the untrained agents. A possible explanation for this could be that the agents are utilizing the return premium associated with higher risk assets (Pedersen, 2015). As real life investors prefer to hold less volatile assets the more volatile assets are less appealing and as such will be cheaper relative to the expected return they generate. Despite this, the agents do all still outperform the untrained model.

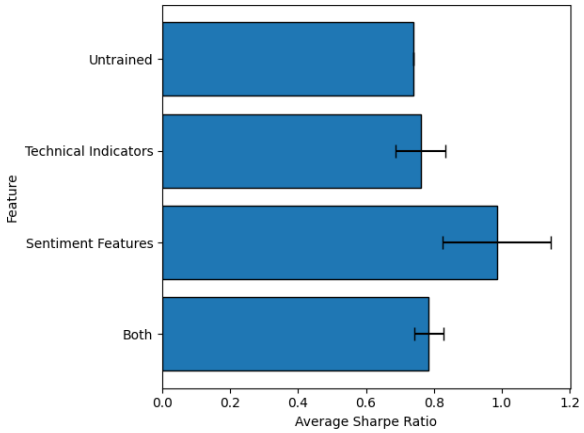


Figure 8.8: Combined Features Refined Sharpe Ratios

8.6 Experiment 5: Model Comparison

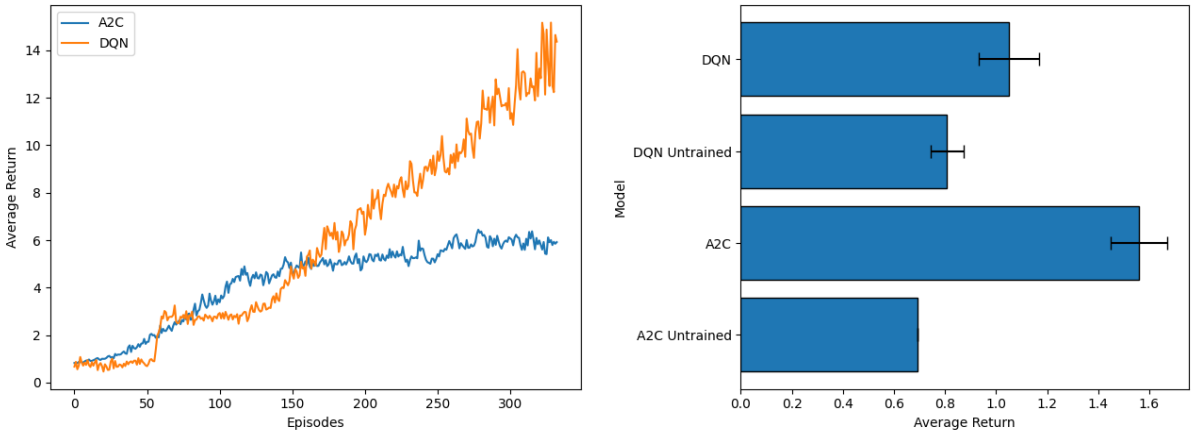
8.6.1 Approach

The final experiment was designed to test Hypothesis 3. For both A2C and DQN 20 agents were tested with a training period of 300,000 timesteps. For comparison, untrained versions of

both models were also tested. Due to the high volatility of the untrained DQN agent 100 of these agents were used to reduce the uncertainty.

8.6.2 Results and Discussion

The results in Figure 8.9 show that during the training period the DQN agents outperformed the A2C agents. This may be explained by overfitting because the discrete nature of the DQN environment facilitates more extreme policies. In the testing both trained agents averaged more than there untrained equivalent agents.



(a) Training (b) Testing

Figure 8.9: Model Comparison

Comparing the two agents we can see that the trained A2C agents have averaged a higher return than the trained DQN agents. To identify the significance of this result we conduct a two-sample T test. First we form a null hypothesis $H_0 : \mu_A \leq \mu_D$ and an alternate hypothesis $H_A : \mu_A > \mu_D$. We then set α to 0.05. Next we calculated the T-value using Equation 8.2. This gives us a value of 3.14. Using a one tailed p value calculator with 19 degrees of freedom this gives us a p value of 0.0027. As this value is less than our α value we can reject the null hypothesis. These results therefore support Hypothesis 3.

$$t = \frac{\overline{x}_A - \overline{x}_D}{\sqrt{\frac{s_A^2}{n_A} + \frac{s_D^2}{n_D}}} \tag{8.2}$$

Figure 8.10 shows the average Sharpe ratios achieved by the agents. Similar to the results displayed in Figure 8.8 the trained agents have performed less well here compared to their overall returns. In this experiment however the A2C agent has managed to outperformed its untrained equivalent.

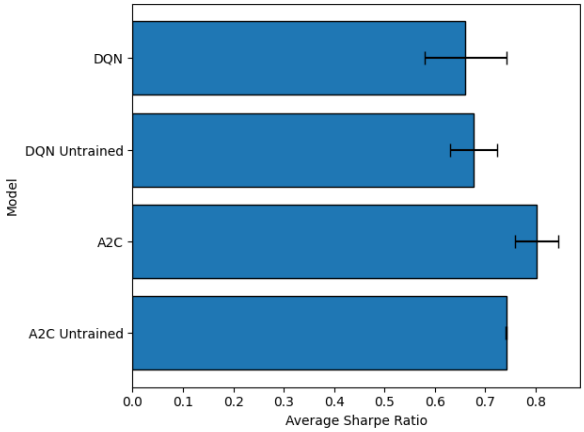


Figure 8.10: Model Comparison Sharpe Ratios

Chapter 9

Conclusion

9.1 Summary

In this project I have sort to achieve the objectives set out in Section 1.2.1. A comprehensive literature and technology review has been conducted covering the areas of stock price prediction, sentiment analysis, deep reinforcement learning, and existing resources. Software has been developed for the purpose of extracting, processing and storing stock data and constructing portfolio allocation environments which can be used by deep reinforcement learning agents. 5 experiments have been conducted with Experiments 2, 3 and 4 addressing Objective 3 and Experiment 5 addressing Objective 4.

9.2 Evaluation

9.2.1 Software

The requirements for the software developed in this project for Objective 2 are set out in Section 6.5.1. For data extraction three types of data were listed that the software should be able to extract on a given stock. Extraction of both historical price and volumes data and historical investment ratings data fully meet these requirements as they can be extracted quickly on any stock. News headlines do also meet their basic requirement, however the extraction of this sort of data with the software does have some limitations. As mentioned in Section 8.1 the quality of news headlines data is substantially lower for less well known stocks. Another limitation is that in order to prevent captcha forms with google news the extraction method needs a great deal of throttling; approximately 15 minutes per stock. This means that for the construction of datasets, such as the ones used for the experimentation portion of this project, a great deal of time is required. Given these limitations the final requirement that the extraction should work for any given stock has only partially been met. Despite this, the approach used did produce substantially better quality data than would have been obtained from any of the other approaches considered.

For data processing there were three requirements. The first requirement for calculation of technical indicators has been met. The second requirements was for daily sentiment values on news headlines. This has been met with three different daily values corresponding to different sentiment analysis resources. The third requirement for daily investment ratings values has also been met with daily values for both ratings and changes in ratings. The data storage

requirements have both been met, with stock objects containing the data easily stored in and retrieved from dill objects using functions.

All portfolio allocation environment requirements have been met. The environment class inherits from gym and implements all necessary methods. The action space of the environment can also be either discrete or continuous. This is decided with a boolean parameter in the constructor. The performance is tracked in terms of overall return, annualized return, and Sharpe ratio.

The requirements for the deep reinforcement learning agents have been met with the use of Stable baselines 3 agents. A limitation to the approach taken here is the reduced customisation of the agents. However, this does make for a fairer comparison of deep reinforcement learning models.

9.2.2 Experimentation

Section 6.6 presents the Hypotheses, the planned agents and the planned comparisons. In Section 8 these experiments were then conducted. Experiment 1 was a parameter search for the A2C model. The results of this were useful as they informed the parameters for the remaining experiments.

To investigate Hypothesis 1, two experiments were conducted. Experiment 2 on the effectiveness of technical indicators found that an agent using technical indicators was able to produce a higher return than an untrained model to a statistically significant degree. Producing a p-value of 0.0035. Experiment 3 on the effectiveness of sentiment features also found that sentiment feature agents produced a higher return than an untrained agent to a statistically significant degree. Producing a p-value of less than 0.0001. These results provide evidence to support Hypothesis 1.

For the investigation of Hypothesis 2, Experiment 4 was conducted. In both of the two sub experiments the agent with both technical analysis and sentiment features did not produce the highest return. These result therefore do not support Hypothesis 2.

Finally, to investigate Hypothesis 3, Experiment 5 was conducted. The results of this experiment showed that the A2C agent did produce higher return than the DQN agent. A two-sample t-test was conducted on this difference and was found to be significant with a p-value of 0.0027. This result therefore provides evidence to support Hypothesis 3. One limitation to this result is that the environment was slightly different for the two agents to facilitate two different types of action. DQN was using discrete actions and A2C was using continuous actions. This meant that if the action space for the DQN agent had been designed differently it may have performed better. Another limitation is that unlike the A2C agent the DQN agent did not receive a parameter search which may have put it at a disadvantage.

There are some limitations to the results of these experiments. Due to the long training time required by these agents a trade off had to be made between the learning timesteps and the number of agents that returns were averaged over. As a consequence of this, some of the results are less significant than they would have been if more agents were used. It also remains unclear how the agents would have performed if they were given more training time. Another limitation comes from the availability of data. Although data is separated into testing and training data the agents were still tested on the same testing data each time. It therefore may be the case that an environment with this data is unusually easy or unusually difficult for

a trained agent to perform well using. Were similar experiments conducted again it may be useful to create environments with variation in the stocks included. For instance, rather than using the top 50 S&P 500 stocks in one environment, 5 environments could be created each with a stock universe of 10 stocks.

9.3 Contributions

The first main contribution of this project is the software. Use cases for this software include extraction, processing and storage of stock data. Data extracted using this software could be used to create a variety of reinforcement learning environments but could also be used in other areas of stock price prediction such as supervised learning. Another use case for this software is the use of the gym portfolio allocation environment which can be used with either discrete or continuous action spaces.

The next main contribution is the experimental results. Parameters for an A2C agent in a portfolio allocation environment have been identified. Statistically significant evidence has been provided that both sentiment features and technical indicators can be used by a deep reinforcement learning agent to achieve higher returns than untrained models. Statistically significant evidence that an A2C agent produced higher returns than a DQN agent in a portfolio allocation environment has also been provided.

A final contribution as a result of the experimentation was the construction of a stock dataset. This dataset does contain prices, volumes, technical indicators, and investment ratings. Most of value however are the news headlines it contains as these take a long time to extract. As previously mentioned this dataset could be useful for many different stock price prediction use cases.

9.4 Further Work

Following this paper there are a number of areas that may be of interest to explore further. In Experiment 2 a comparison is made between the different technical indicators. The focus in this experiment however was the effectiveness of an agent using all of these indicators. It may be of interest to conduct an experiment comparing a wide variety of technical indicators. It could also be of interest to further investigate the use of individual sentiment features. For instance, a more focused experiment on the effectiveness of individual sentiment analysis tools or a more in depth comparison of Rating and Change in Rating features. It could also be worth investigating different values for use as the sentiment decay parameter λ. This could also have interesting implications for the decay of sentiment importance in the stock market more generally. Although fundamental indicators, such as those presented in Table 2.2, were not included in the experimentation for this project, it could be informative to compare their effectiveness if an appropriate dataset was constructed.

Another direction for future work could be to test another deep reinforcement learning model such as DDPG. This would help to identify the differences between critic-only and actor-critic models in a portfolio allocation environment.

To further understand the implications of these results it could be useful to see how agents perform when the reward signal is based on risk adjusted return. There are many different ways that this could be implemented. One approach for instance, could be to calculate the

the change in Sharpe ratio over the last 100 days from one timestep to the next. This could help to more closely align a model to achieve real world portfolio management objectives.

Bibliography

Aroussi, R., 2019. yfinance documentation [Online]. Available from: `https://pypi.org/project/yfinance/`.

Azhikodan, A.R., Bhat, A.G. and Jadhav, M.V., 2019. Stock trading bot using deep reinforcement learning. *Innovations in computer science and engineering*. Springer, pp.41–49.

Barber, B.M., Lehavy, R. and Trueman, B., 2010. Ratings changes, ratings levels, and the predictive value of analysts' recommendations. *Financial management*, 39(2), pp.533–553.

Bettman, J.L., Sault, S.J. and Schultz, E.L., 2009. Fundamental and technical analysis: substitutes or complements? *Accounting & finance*, 49(1), pp.21–36.

Beyaz, E., Tekiner, F., Zeng, X.j. and Keane, J., 2018. Comparing technical and fundamental indicators in stock price forecasting. *2018 ieee 20th international conference on high performance computing and communications; ieee 16th international conference on smart city; ieee 4th international conference on data science and systems (hpcc/smartcity/dss)*. IEEE, pp.1607–1613.

Bollen, J., Mao, H. and Zeng, X., 2011. Twitter mood predicts the stock market. *Journal of computational science*, 2(1), pp.1–8.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. Openai gym. *arxiv preprint arxiv:1606.01540*.

Button, M., 2019. How to scrape stock upgrades and downgrades from yahoo finance. Available from: `https://www.mattbutton.com/how-to-scrape-stock-upgrades-and-downgrades-from-yahoo-finance/`.

Carta, S., Corriga, A., Ferreira, A., Podda, A.S. and Recupero, D.R., 2021. A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning. *Applied intelligence*, 51(2), pp.889–905.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arxiv preprint arxiv:1412.3555*.

Dang, M. and Duong, D., 2016. Improvement methods for stock market prediction using financial news articles. *2016 3rd national foundation for science and technology development conference on information and computer science (nics)*. IEEE, pp.125–129.

Feng, Y., 2019. Create a customized gym environment for star craft 2. Available from: `https://towardsdatascience.com/create-a-customized-gym-environment-for-star-craft-2-8558d301131f`.

Fischer, T.G., 2018. *Reinforcement learning in financial markets-a survey*. FAU Discussion Papers in Economics.

FKnol, 2021. Market cap of s&p 500 index companies 2021. Available from: `https://fknol.com/list/market-cap-sp-500-index-companies.php` [Accessed 2021-07-01].

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G. and Pineau, J., 2018. An introduction to deep reinforcement learning. *arxiv preprint arxiv:1811.12560*.

Hu, Z., Zhao, Y. and Khushi, M., 2021. A survey of forex and stock price prediction using deep learning. *Applied system innovation*, 4(1), p.9.

Huang, Y., 2018. Comparision between ddpg and a2c. Available from: `https://yodahuang.github.io/articles/DDPG-vs-A2C/`.

Hussein, D.M.E.D.M., 2018. A survey on sentiment analysis challenges. *Journal of king saud university-engineering sciences*, 30(4), pp.330–338.

Hutto, C. and Gilbert, E., 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the international aaai conference on web and social media*. vol. 8.

Jeong, Y., Kim, S. and Yoon, B., 2018. An algorithm for supporting decision making in stock investment through opinion mining and machine learning. *2018 portland international conference on management of engineering and technology (picmet)*. IEEE, pp.1–10.

Jia, W., Chen, W., Xiong, L. and Hongyong, S., 2019. Quantitative trading on stock market based on deep reinforcement learning. *2019 international joint conference on neural networks (ijcnn)*. IEEE, pp.1–8.

Kaur, S., 2017. Algorithmic trading using sentiment analysis and reinforcement learning. *positions*.

Kirange, M.D. and Deshmukh, R.R., 2016. Sentiment analysis of news headlines for stock price prediction. *Compusoft: An international journal of advanced computer technology*, 5(3).

Koratamaddi, P., Wadhwani, K., Gupta, M. and Sanjeevi, S.G., 2021. Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation. *Engineering science and technology, an international journal*.

Krantz, M., 2016. *Fundamental analysis for dummies*. John Wiley & Sons.

Larsen, J.I., 2010. *Predicting stock prices using technical analysis and machine learning*. Master's thesis. Institutt for datateknikk og informasjonsvitenskap.

Li, X., Li, Y., Zhan, Y. and Liu, X.Y., 2019. Optimistic bull or pessimistic bear: Adaptive deep reinforcement learning for stock portfolio allocation. *arxiv preprint arxiv:1907.01503*.

Li, X., Xie, H., Chen, L., Wang, J. and Deng, X., 2014. News impact on stock price return via sentiment analysis. *Knowledge-based systems*, 69, pp.14–23.

Li, Y., Ni, P. and Chang, V., 2019a. Application of deep reinforcement learning in stock trading strategies and stock forecasting. *Computing*, pp.1–18.

Li, Y., Ni, P. and Chang, V., 2019b. An empirical research on the investment strategy of stock market based on deep reinforcement learning model. *Complexis*. pp.52–58.

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. *arxiv preprint arxiv:1509.02971*.

Liu, X.Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B. and Wang, C.D., 2020. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance. *arxiv preprint arxiv:2011.09607*.

MacroTrends, 2021. 10 year treasury rate - 54 year historical chart. Available from: `https://www.macrotrends.net/2016/10-year-treasury-bond-rate-yield-chart` [Accessed 2021-07-01].

Medhat, W., Hassan, A. and Korashy, H., 2014. Sentiment analysis algorithms and applications: A survey. *Ain shams engineering journal*, 5(4), pp.1093–1113.

Mehlig, B., 2019. Artificial neural networks. *arxiv preprint arxiv:1901.05639*.

Meng, T.L. and Khushi, M., 2019. Reinforcement learning in financial markets. *Data*, 4(3), p.110.

Michaux, J., 2019. Off-policy actor-critic algorithms. Available from: `https://jmichaux.github.io/week4b/`.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*. PMLR, pp.1928–1937.

Nam, K. and Seong, N., 2019. Financial news-based stock movement prediction using causality analysis of influence in the korean stock market. *Decision support systems*, 117, pp.100–112.

Nan, A., Perumal, A. and Zaiane, O.R., 2020. Sentiment and knowledge based algorithmic trading with deep reinforcement learning. *arxiv preprint arxiv:2001.09403*.

Nguyen, T.H., Shirai, K. and Velcin, J., 2015. Sentiment analysis on social media for stock movement prediction. *Expert systems with applications*, 42(24), pp.9603–9611.

Obthong, M., Tantisantiwong, N., Jeamwatthanachai, W. and Wills, G., 2020. A survey on machine learning for stock price prediction: algorithms and techniques. *Proceedings of the 2nd international conference on finance, economics, management and it business*.

Oriani, F.B. and Coelho, G.P., 2016. Evaluating the impact of technical indicators on stock forecasting. *2016 ieee symposium series on computational intelligence (ssci)*. IEEE, pp.1–8.

Pedersen, L.H., 2015. *Efficiently inefficient*. Princeton University Press.

PSF, 2011. requests. `https://github.com/psf/requests`.

Quantopian, 2015. Pyfolio. `https://github.com/quantopian/pyfolio`.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A. and Dormann, N., 2019. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`.

Richardson, L., 2007. Beautiful soup documentation. *April*.

Rockefeller, B., 2019. *Technical analysis for dummies*. John Wiley & Sons.

SeleniumHQ, 2018. selenium. `https://github.com/SeleniumHQ/selenium`.

Shah, D., Isah, H. and Zulkernine, F., 2018. Predicting the effects of news sentiments on the stock market. *2018 ieee international conference on big data (big data)*. IEEE, pp.4705–4708.

Shin, H.G., Ra, I. and Choi, Y.H., 2019. A deep multimodal reinforcement learning system combined with cnn and lstm for stock trading. *2019 international conference on information and communication technology convergence (ictc)*. IEEE, pp.7–11.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. et al., 2019. Huggingface's transformers: State-of-the-art natural language processing. *arxiv preprint arxiv:1910.03771*.

Wu, X., Chen, H., Wang, J., Troiano, L., Loia, V. and Fujita, H., 2020. Adaptive stock trading strategies with deep reinforcement learning methods. *Information sciences*, 538, pp.142–158.

Zhuang, C., 2020. stockstats. `https://github.com/jealous/stockstats`.

Appendix A

User Guide

A.1 Overview

The key features of this software are listed below.

- Class for stock data extraction, processing and storage.
- Portfolio allocation environment for deep reinforcement learning with either discrete or continuous action spaces.
- Dataset for the top 50 S&P 500 stocks including price data, volumes data and news headlines; as well as technical indicators and sentiment scores derived from this data.

A.2 Installation

To install the project using pip with python the following steps should be followed.

A.2.1 Creating Virtual Environment

First pip install virtualenv by running:

```
pip install virtualenv
```

Next, while in the folder for this project create a new virtual environment using:

```
python -m venv venv
```

A.2.2 Activating Virtual Environment

From the project folder the virtual environment can now be activated. If using Windows this can be done by running:

```
venv\Scripts\activate
```

Or, if using Linux or Mac running:

```
source venv/bin/activate
```

A.2.3 Installing Dependencies

Finally, with this new environment activated the dependencies for the project can be installed by running:

```
pip install -r requirements.txt
```

A.2.4 Downloading Chrome Web Driver (Optional)

This step is required in order to extract news headlines on given stocks, which is a part of the `extract_and_calculate_all` method but is not included in the `extract_and_calculate_basic` method.

For this to work Chrome will already need to be installed. If this is the case then go to `https://chromedriver.chromium.org/downloads` and download the relevant file for your operating system and version of Chrome. Put this downloaded file in the "other chromedriver" folder. When run the code should recognise this file, providing its filename starts with "chromedriver"

A.3 Usage

To run files from this repository first activate the virtual environment as described in the installation.

A.3.1 Data Extraction and Processing

To extract and process data first create a `Stock` object in the `main.py` file. Data can then be extracted from the given stock and processed using one of the extraction methods such as `extract_and_calculate_basic`. To get the DataFrame from this object the attribute can simply be assigned to a variable. An example of this process is provided below.

```
s = Stock('Apple', 'AAPL')
s.extract_and_calculate_basic()
df = s.df
```

A.3.2 Data Storage and Retrieval

The `storage.py` file includes some functions for saving and loading stock objects to and from dill files. To save and retrieve stocks the `save_dill_object` and `retrieve_dill_object` functions can be used. For example.

```
save_dill_object(s, 'data/my_stock.dill')
retrived_stock = retrieve_dill_object('data/my_stock.dill')
```

Another function has also been provided to retrieve all stocks in a given folder called `retrieve_stocks_from_folder`. An example of using this function to extract the provided dataset, the top 50 S&P 500 stocks, is given below.

```
list_of_stocks = retrieve_stocks_from_folder('data/snp_50_stocks_full')
```

A.3.3 Portfolio Allocation Environment

A portfolio allocation environment for reinforcement learning can be created by instantiating an object of the `PortfolioAllocationEnvironment` class. The constructor of this class takes three main parameters. `stocks` should be provided as a list of stock DataFrames, which should be of the same length and over the same time period. `state_attributes` takes a list of attributes, as strings, that will be used to represent the state space. To decide which of these to use check the column names of the stock DataFrames. The final parameter is the boolean `discrete`, which determines how the action space will be structured. If this is set to true the action space is discrete, otherwise it will be continuous. A full example of creating an environment is provided below.

```
stocks = retrieve_stocks_from_folder("data/snp_50_stocks_full")
dfs = [s.df for s in stocks]
attributes = ['normalized_rsi', 'std_devs_out', 'relative_vol', '
   hf_google_articles_score', 'vader_google_articles_score', '
   ranking_score']

env = PortfolioAllocationEnvironment(dfs, attributes, discrete=True)
```

A.3.4 Using Environment

To use this environment with a deep reinforcement learning agent the data should be split into a testing and training set and two different environments can be constructed. An agent can then be trained on the first environment and tested on the second environment. An example of this process is provided below using an agent from Stable Baselines 3.

```
from stable_baselines3 import A2C

# Create and Train Agent
attributes = ['std_devs_out', 'vader_google_articles_score']
train_dfs = [s.df.loc[100:1000] for s in stocks[:]]
train_env = PortfolioAllocationEnvironment(
        train_dfs, attributes, discrete=False)
model = A2C('MlpPolicy', train_env, verbose=0, learning_rate=0.0005,
   gamma=0)
model.learn(total_timesteps=100_000)

# Test Agent
test_dfs = [s.df.loc[1000:] for s in stocks[:]]
test_env = PortfolioAllocationEnvironment(
        test_dfs, attributes, discrete=False)
obs = test_env.reset()
while True:
    action, _state = model.predict(obs, deterministic=True)
    obs, reward, done, info = test_env.step(action)
    if done:
        break
print(test_env.annualized_return)
```

Appendix B

Code

B.1 Environment Class

This is the environment class that can be used by deep reinforcement learning agents. It follows the Gym framework and can be instantiated with either a discrete or continuous action space. It also tracks the performance of all the episodes run on it.

```
from stock import *
import random

import numpy as np
import pandas as pd
from gym.utils import seeding
import gym
from gym import spaces
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt


class PortfolioAllocationEnvironment(gym.Env):

    def __init__(self, stocks, state_attributes, discrete=False):
        self.check_arguments_valid(stocks, state_attributes)
        self.starting_value = 1_000_000
        self.stocks = [s.reset_index() for s in stocks]
        self.state_attributes = state_attributes
        self.final_index = len(stocks[0]) - 1
        self.final_values = []
        self.annualized_returns = []
        self.sharpe_ratios = []

        self.discrete = discrete
        if self.discrete:
            self.action_space = spaces.Discrete(len(self.stocks))
        else:
            self.action_space = spaces.Box(low=-np.inf, high=np.inf,
    shape=(len(self.stocks),))
        self.observation_space = spaces.Box(low=-np.inf, high=np.inf,
    shape=(len(self.state_attributes), len(self.stocks)))

```

```
33      def reset(self):
34          self.date_index = 0
35          self.portfolio_value = self.starting_value
36          self.state = self.get_state()
37          self.terminal = False
38          self.return_memory = [0]
39          self.value_memory = [self.portfolio_value]
40          self.date_memory = [self.stocks[0].loc[self.date_index, 'date'
    ]]
41          self.actions_memory = [np.array([1 for _ in range(len(self.
    stocks))])]
42          self.reward_memory = []
43          return self.state
44
45      def step(self, action):
46          if self.terminal:
47              raise Exception("Environment already in terminal state")
48
49          previous_index = self.date_index
50          self.date_index += 1
51
52          if self.discrete:
53              s = self.stocks[action]
54              previous_close = s.loc[previous_index, 'close']
55              new_close = s.loc[self.date_index, 'close']
56              portfolio_return = (new_close / previous_close) - 1
57          else:
58              weights = self.softmax_normalization(action)
59              previous_closes = np.array([s.loc[previous_index, 'close']
    for s in self.stocks])
60              new_closes = np.array([s.loc[self.date_index, 'close'] for
    s in self.stocks])
61              portfolio_return = sum(((new_closes / previous_closes) - 1)
     * weights)
62
63          change_in_value = self.portfolio_value * portfolio_return
64          self.portfolio_value *= (1 + portfolio_return)
65
66          # Memory management
67          self.return_memory.append(portfolio_return)
68          self.value_memory.append(self.portfolio_value)
69          self.date_memory.append(self.stocks[0].loc[self.date_index, '
    date'])
70          self.actions_memory.append(action)
71
72          self.state = self.get_state()
73          self.reward = change_in_value
74          self.reward_memory.append(self.reward)
75
76          if self.date_index >= self.final_index:
77              self.terminal = True
78              # Performance metrics
79              self.final_values.append(self.portfolio_value)
80              days = self.final_index + 1
81              years = days / TRADING_DAYS
82              total_return = (self.portfolio_value / self.starting_value)
     - 1
```

```
83              self.annualized_return = ((total_return + 1) ** (1 / years)
    ) - 1
84              annualized_volatility = np.std(self.return_memory) * math.
    sqrt(TRADING_DAYS)
85              self.sharpe_ratio = (self.annualized_return - BASE_RATE) /
    annualized_volatility
86              self.sharpe_ratios.append(self.sharpe_ratio)
87              self.annualized_returns.append(self.annualized_return)
88
89          return self.state, self.reward, self.terminal, {}
90
91      def get_state(self):
92          return np.array([[s.loc[self.date_index, a] for s in self.
    stocks] for a in self.state_attributes])
93
94      @staticmethod
95      def softmax_normalization(actions):
96          a = actions.astype('float64')
97          return np.exp(actions) /np.sum(np.exp(a))
98
99      @staticmethod
100     def check_arguments_valid(stocks, state_attributes):
101         for stock in stocks:
102             if len(stock) != len(stocks[0]):
103                 raise ValueError("Length of stock DataFrames provided
    do not match")
104             for a in state_attributes:
105                 if a not in stock.columns:
106                     raise ValueError(f"State attribute {a} not
    available in at least one stock.")
```

B.2 Stock Class

This class is used primarily for the extraction and processing of stock data. It can extract price and volumes data as well as investment ratings and news headlines. This data can then be processed to produce technical indicators and daily sentiment values from both news headlines and investment ratings.

```
1  from constants import *
2  from storage import *
3  # General
4  from datetime import datetime
5  import numpy as np
6  import pandas as pd
7  import math
8  import time
9  # Scraping
10 import requests
11 from bs4 import BeautifulSoup
12 import threading
13 import re
14 import json
15 import selenium
16 from selenium import webdriver
17 from selenium.webdriver.common.by import By
18 from selenium.webdriver.support.ui import WebDriverWait
```

```
from selenium.webdriver.support import expected_conditions as EC
# Sentiment
from transformers import pipeline
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer


class Stock:
    def __init__(self, name, code, ic_name="", start_date="2014-01-01", end_date="2021-01-01", df=pd.DataFrame(), search_term=None, driver=None):
        self.name = name
        self.code = code
        self.ic_name = ic_name
        self.start_date = self._iso_to_datetime(start_date)
        self.end_date = self._iso_to_datetime(end_date)
        self.df = self._initialize_df() if df.empty else df
        self.search_term = name if search_term == None else search_term
        self.driver = driver

        # Parameters
        self.rs_decay = 0.9
        self.ss_decay = 0.9

        # Links
        self.investing_link =f"https://uk.investing.com/equities/{self.ic_name}"
        self.yahoo_link = f"https://uk.finance.yahoo.com/quote/{self.code}"

    def __str__(self):
        rep = f"{self.code} - {self.name}\n{self.df.describe()}"
        return rep

    # Data Extraction Methods
    def _initialize_df(self):
        """Extract price data and use this to create a new dataframe for the stock"""
        def _convert_type(value):
            data_types = [int, float, str]
            for d in data_types:
                try:
                    return d(value)
                except ValueError:
                    pass
        data_points = []
        period1 = str(int(self.start_date.timestamp()))
        period2 = str(int(self.end_date.timestamp()))
        interval = "1d"
        file_link = (f"https://query1.finance.yahoo.com/v7/finance/download/"
            f"{self.code}?period1={period1}&period2={period2}&interval={interval}")
        request = requests.get(file_link, headers=STANDARD_HEADERS)
        if request.status_code == 401:
            raise Exception("Yahoo finance rejected request")
        content = str(request.content).replace("'", "").split("\\n")
        # cols = content[0].split(",")
```

```
70          for i in range(1, len(content)):
71              new_point = [_convert_type(d) for d in [self.code] +
    content[i].split(",")]
72              new_point[1] = self._iso_to_datetime(new_point[1])
73              data_points.append(new_point)
74          if len(data_points) == 0:
75              raise Exception("No data was retrieved by the extraction
    function")
76          return pd.DataFrame(data=data_points, columns=['tic', 'date', '
    open', 'high', 'low', 'close', 'adj_close', 'volume'])
77
78      def extract_investment_ranking_data(self):
79          url = f"https://uk.finance.yahoo.com/quote/{self.code}/analysis
    "
80          page = requests.get(url, headers=STANDARD_HEADERS)
81          if page.status_code == 401:
82              raise Exception("Yahoo finance rejected request")
83
84          # 5 lines below adapted from M Button, 2019
85          # https://www.mattbutton.com/
86          #   how-to-scrape-stock-upgrades-and-downgrades-from-yahoo-
    finance/
87          soup = BeautifulSoup(page.content,'html.parser')
88          script = soup.find('script', text=re.compile(r'root\.App\.main'
    ))
89          json_text = re.search(r'^\s*root\.App\.main\s*=\s*({.*?})\s*;\s
    *$', script.string, flags=re.MULTILINE).group(1)
90          data = json.loads(json_text)
91          rankings_scraped = data['context']['dispatcher']['stores']\
92              ['QuoteSummaryStore']['upgradeDowngradeHistory']['history']
93
94          rankings_dict = dict()
95          for r in rankings_scraped:
96              investment_ranking = {'action':r['action'], 'from':r['
    fromGrade'], 'to': r['toGrade']}
97              iso = datetime.fromtimestamp(r['epochGradeDate']).strftime(
    '%Y-%m-%d')
98              date = self._iso_to_datetime(iso)
99              if date in rankings_dict:
100                 rankings_dict[date].append(investment_ranking)
101             else:
102                 rankings_dict[date] = [investment_ranking]
103         # Rankings by date list
104         rankings_by_date_list = []
105         ranking_scores = []
106         previous_rank_score = 0
107         change_scores = []
108         previous_change_score = 0
109         for i in range(len(self.df)):
110             if self.df.iloc[i]['date'] in rankings_dict:
111                 rankings = rankings_dict[self.df.iloc[i]['date']]
112                 ranking_values = [RANKING_VALUES[r['to']] if r['to'] in
     RANKING_VALUES else 0 for r in rankings]
113                 for r in rankings:
114                     if r['to'] not in RANKING_VALUES:
115                         print(r)
116                 ranking_score = (self.rs_decay * previous_rank_score) +
    sum(ranking_values)
```

```
                change_values = [CHANGE_VALUES[r['action']] for r in
rankings]
                change_score = (self.rs_decay * previous_change_score)
+ sum(change_values)
            else:
                rankings = []
                ranking_score = self.rs_decay * previous_rank_score
                change_score = self.rs_decay * previous_change_score
            previous_rank_score = ranking_score
            previous_change_score = change_score
            rankings_by_date_list.append(rankings)
            ranking_scores.append(ranking_score)
            change_scores.append(change_score)

        self.df['rankings'] = rankings_by_date_list
        self.df['ranking_score'] = ranking_scores
        self.df['ranking_change_score'] = change_scores

    def extract_news_data(self, google=True, investing=False, threads
=5, verbose=False):

        def _extract_from_investing_page(articles_dict, n, i, valid):

            def _convert_investing_date(news_date):
                # Converts format of date e.g. Apr 14, 2021 ->
2021-04-14
                if "ago" in news_date:
                    return datetime.today().strftime('%Y-%m-%d')

                date_components = news_date.replace(",", "").split(" ")
                return (f"{date_components[2]}-"
                    f"{MONTHS[date_components[0].lower()]}-{
date_components[1]}")

            url = f"https://uk.investing.com/equities/{self.ic_name}-
news/{str(n)}"
            page = requests.get(url, headers=STANDARD_HEADERS)
            soup = BeautifulSoup(page.content,'html.parser')

            # Check you haven't looped
            p_number = soup.find('div', id='paginationWrap').find('a',
class_='pagination selected').text

            if n != int(p_number):
                valid[i] = False
            else:
                articles_section = soup.find('section', id='leftColumn'
)
                articles = articles_section.find_all('article')
                for article in articles:
                    d = article.find('div')
                    a = d.find('a')
                    details_sec = d.find(class_= 'articleDetails')
                    details = details_sec.find_all('span')

                    title = a.text
                    link = a['href']
                    author = details[0].text[3:]
```

```
167                     date = self._iso_to_datetime(
    _convert_investing_date(details[1].text[3:]))
168 
169                     new_article = {'title': title, 'link': link, '
    author': author}
170 
171                     if date in articles_dict:
172                         articles_dict[date].append(new_article)
173                     else:
174                         articles_dict[date] = [new_article]
175 
176                 valid[i] = True
177 
178         if investing:
179             if verbose: print("Extracting Investing News")
180             # Populate a dictionary with scraped articles
181             investing_articles_dict = dict()
182             # Use threading to request pages and extract articles
183             start_page = 1
184             searching = True
185             while searching:
186                 if verbose: print(f"{start_page} - {start_page +
    threads}")
187                 threads_list = []
188                 valid = [None] * threads
189                 for n in range(start_page, start_page + threads):
190                     t = threading.Thread(target=
    _extract_from_investing_page, args=(investing_articles_dict, n, n -
    1 - start_page, valid))
191                     t.start()
192                     threads_list.append(t)
193                 for t in threads_list:
194                     t.join()
195                 if not all(valid):
196                     searching = False
197                 else:
198                     start_page += threads
199 
200             investing_articles = []
201             for i in range(len(self.df)):
202                 if self.df.loc[i,'date'] in investing_articles_dict:
203                     investing_articles.append(investing_articles_dict[
    self.df.loc[i,'date']])
204                 else:
205                     investing_articles.append([])
206 
207             # Add articles to DataFrame
208             self.df['investing_articles'] = investing_articles
209 
210         def _extract_from_google_news_searchs(articles_dict, driver,
    pages_per_range=10): #, from_date, to_date, max_pages=1000
211 
212             def _convert_google_date(date_string):
213                 # Converts date in format e.g. 22 Sept 2020 ->
    2020-09-22
214                 date_components = date_string.split(" ")
215                 return (f"{date_components[2]}-"
```

```
216                     f"{MONTHS[date_components[1].lower()[:3]]}-{
    date_components[0]}")
217
218             def _check_for_captcha(driver):
219                 try:
220                     driver.find_element_by_xpath('//*[@id="captcha-form
    "]')
221                     time.sleep(2)
222                     print("\nThere appears to be Captcha form. Complete
     this and then press enter.")
223                     input()
224                 except selenium.common.exceptions.
    NoSuchElementException as e:
225                     pass
226             def _return_element(driver, xpath):
227                 WebDriverWait(driver, 5).until(EC.
    presence_of_element_located((By.XPATH, xpath)))
228                 return driver.find_element_by_xpath(xpath)
229
230             def _change_date_range(driver, from_date, to_date):
231                 _return_element(driver, '//*[@id="hdtbMenus"]/span[2]/g
    -popup/div[1]').click()
232                 _return_element(driver, '//*[@id="1b"]/div/g-menu/g-
    menu-item[8]').click()
233                 from_date_box = _return_element(driver, '//*[@id="
    OouJcb"]')
234                 from_date_box.clear()
235                 from_date_box.send_keys(from_date)
236                 to_date_box = _return_element(driver, '//*[@id="rzG2be
    "]')
237                 to_date_box.clear()
238                 to_date_box.send_keys(to_date)
239                 _return_element(driver, '//*[@id="T3kYXe"]/g-button').
    click()
240
241             def _change_search_term(driver, search):
242                 search_box = _return_element(driver, '//*[@id="lst-ib"]
    ')
243                 search_box.clear()
244                 search_box.send_keys(search)
245                 # search_box.submit()
246                 _return_element(driver, '//*[@id="mKlEF"]').click()
247
248             # Change search term
249             _change_search_term(driver, self.search_term)
250             # Check to see if there is a Capcha form that needs to be
    completed
251             _check_for_captcha(driver)
252
253             earliest_year = self.df.loc[0,'date'].year
254             latest_year = self.df.loc[len(self.df) - 1,'date'].year
255             for y in range(earliest_year, latest_year + 1):
256                 if str(y) == "2018": time.sleep(60 * 7) # Mid
    extraction throttle
257                 from_date = f"01/01/{y}"
258                 to_date = f"12/31/{y}"
259                 _change_date_range(driver, from_date, to_date)
260                 # Iterate through pages
```

```
261                 for i in range(pages_per_range):
262                     _check_for_captcha(driver)
263                     # Extract articles from page
264                     page = driver.page_source
265                     soup = BeautifulSoup(page,'html.parser')
266                     articles_div = soup.find('div', id='rso')
267                     if articles_div == None:
268                         pass
269                     else:
270                         a_divs = articles_div.find_all('div', class_="
    dbsr")
271                         for a_div in a_divs:
272                             article = dict()
273                             article['title'] = a_div.find('div', role="
    heading").text
274                             article['link'] = a_div.find('a')['href']
275                             try:
276                                 iso = _convert_google_date(a_div.find('
    span', class_="WG9SHc").text)
277                                 date = self._iso_to_datetime(iso)
278                                 if date in articles_dict:
279                                     articles_dict[date].append(article)
280                                 else:
281                                     articles_dict[date] = [article]
282                             except KeyError:
283                                 pass
284
285                     # Move to next page
286                     WebDriverWait(driver, 5).until(EC.
    presence_of_element_located((By.XPATH, '//*[@id="pnnext"]')))
287                     next_button = driver.find_element_by_xpath('//*[@id
    ="pnnext"]')
288                     next_button.click()
289
290         if google:
291             if verbose: print("Extracting Google News")
292             if self.driver == None:
293                 driver_local = True
294                 driver = self.get_google_news_driver()
295             else:
296                 driver_local = False
297                 driver = self.driver
298
299             # Extract news headlines from google news
300             google_articles_dict = dict()
301             _extract_from_google_news_searchs(google_articles_dict,
    driver)
302
303             # Quit driver when finished
304             if driver_local: driver.quit()
305             self.driver = None
306
307             # Create dated list from dictionary
308             google_articles = []
309             for i in range(len(self.df)):
310                 if self.df.loc[i,'date'] in google_articles_dict:
311                     google_articles.append(google_articles_dict[self.df
    .loc[i,'date']])
```

```
312                 else:
313                     google_articles.append([])
314 
315             # Add articles to DataFrame
316             self.df['google_articles'] = google_articles
317 
318     # Data Processing Methods
319     def calculate_technical_indicators(self):
320         """Calculates technical indicators and adds them to the
    DataFrame."""
321         # MACD
322         smoothing = 2
323         ema12 = [None for _ in range(11)]
324         ema26 = [None for _ in range(25)]
325         macd = [None for _ in range(25)]
326         ema12.append(sum(self.df.iloc[:12]['close'])/12)
327         ema26.append(sum(self.df.iloc[:26]['close'])/26)
328         for i in range(12, len(self.df)):
329             if i > 11: # EMA 12
330                 ema12.append((self.df.iloc[i]['close'] * (smoothing /
    13)) + (ema12[i-1] * (1 - (smoothing / 13))))
331             if i > 25: # EMA 26
332                 ema26.append((self.df.iloc[i]['close'] * (smoothing /
    27)) + (ema26[i-1] * (1 - (smoothing / 27))))
333             if i > 24: # MACD
334                 macd.append(ema12[i] - ema26[i])
335         signal_line = [None for _ in range(33)]
336         signal_line.append(sum(macd[25:34]) / 9)
337         for i in range(34, len(self.df)):
338             signal_line.append((macd[i] * (smoothing / 10)) + (
    signal_line[i-1] * (1 - (smoothing / 10))))
339 
340         # RSI
341         # Calculating close changes
342         close_change = [None]
343         for i in range(1, len(self.df)):
344             close_change.append(self.df.iloc[i]['close'] - self.df.iloc
    [i-1]['close'])
345         up_sma14 = [None for _ in range(14)]
346         down_sma14 = [None for _ in range(14)]
347         rsi = [None for _ in range(14)]
348         normalized_rsi = [None for _ in range(14)]
349         # Up and Down SMA 14
350         for i in range(14, len(self.df)):
351             # ups = list(filter(lambda x: x > 0, close_change[i-13:i
    +1]))
352             ups = [x for x in close_change[i-13:i+1] if x > 0]
353             up_sma14.append(1e-8 if len(ups) == 0 else sum(ups) / 14)
354             # downs = list(map(lambda x:abs(x), filter(lambda x: x < 0,
     close_change[i-13:i+1])))
355             downs = [abs(x) for x in close_change[i-13:i+1] if x < 0]
356             down_sma14.append(1e-8 if len(downs) == 0 else sum(downs) /
     14)
357             rsi.append(100 - (100 / (1 + (up_sma14[i] / down_sma14[i]))
    ))
358             normalized_rsi.append(rsi[i] / 100)
359 
360         # Bollinger Bands
```

```
361         sma20 = [None for _ in range(19)]
362         std_dev20 = [None for _ in range(19)]
363         bb_upper = [None for _ in range(19)]
364         bb_lower = [None for _ in range(19)]
365         std_devs_out = [None for _ in range(19)]
366         for i in range(19, len(self.df)):
367             sma20.append(sum(self.df.iloc[i-19:i+1]['close']) / 20)
368             std_dev20.append(math.sqrt(sum([(d - sma20[i])**2 for d in
    self.df.loc[:,'close'].iloc[i-19:i+1]]) / 20))
369             bb_upper.append(sma20[i] + (2 * std_dev20[i]))
370             bb_lower.append(sma20[i] - (2 * std_dev20[i]))
371             std_devs_out.append((self.df.iloc[i]['close'] - sma20[i]) /
     std_dev20[i])
372 
373         # Relative Volume
374         vol_sma60 = [None for _ in range(59)]
375         relative_vol = [None for _ in range(59)]
376         for i in range(59, len(self.df)):
377             vol_sma60.append(sum(self.df.loc[i-59:i,'volume']) / 60)
378             relative_vol.append(self.df.iloc[i]['volume'] / vol_sma60[i
    ])
379 
380         generated_lists = {
381             'ema12': ema12, 'ema26': ema26, 'macd': macd, 'signal_line'
    : signal_line, 'close_change': close_change,
382             'up_sma14': up_sma14, 'down_sma14': down_sma14, 'rsi': rsi,
     'normalized_rsi': normalized_rsi, 'sma20': sma20,
383             'std_dev20': std_dev20, 'bb_upper': bb_upper, 'bb_lower':
    bb_lower, 'std_devs_out': std_devs_out,
384             'vol_sma60': vol_sma60, 'relative_vol': relative_vol}
385 
386         # Check that the list lengths are correct
387         if not all([len(self.df) == len(l) for l in generated_lists.
    values()]):
388             raise Exception("One of the lists in technical indicators
    calculation was wrong length")
389 
390         # Add technical indicator lists to the dataframe
391         for key, value in generated_lists.items():
392             self.df[key] = value
393 
394     def calculate_news_sentiment(self, hugging_face=True, text_blob=
    False, vader=False, verbose=False):
395         """Use various libraries to extract sentiment from all
    available news articles"""
396         # Check which sources we have extracted from
397         possible_sources = ["google_articles", "investing_articles"]
398         available_sources = []
399         for source in possible_sources:
400             if source in self.df:
401                 available_sources.append(source)
402         # Extract sentiment from available sources
403         for source in available_sources:
404             if hugging_face:
405                 if verbose: print("Hugging face")
406                 classifier = pipeline("sentiment-analysis")
407                 for i in range(len(self.df)):
408                     for a in self.df.loc[i, source]:
```

```
409                         a['hugging_face'] = classifier(a['title'])[0]
410                         if verbose: print(a['hugging_face'])
411 
412                 if verbose: print("Calculating Hugging face scores")
413                 previous = 0
414                 scores = []
415                 for i in range(len(self.df)):
416                     if len(self.df.loc[i, source]) > 0:
417                         values = [HF_LABEL_VALUES[a['hugging_face']['
    label']] for a in self.df.loc[i, source]]
418                         score = (self.ss_decay * previous) + sum(values
    )
419                     else:
420                         score = (self.ss_decay * previous)
421                     scores.append(score)
422                     previous = score
423                 self.df[f'hf_{source}_score'] = scores
424 
425             if text_blob:
426                 if verbose: print("Text blob")
427                 for i in range(len(self.df)):
428                     for a in self.df.loc[i, source]:
429                         title_sentiment = TextBlob(a['title']).
    sentiment
430                         a['text_blob'] = {"polarity": title_sentiment.
    polarity, "subjectivity": title_sentiment.subjectivity}
431 
432                 if verbose: print("Calculating Text blob scores")
433                 previous = 0
434                 scores = []
435                 for i in range(len(self.df)):
436                     if len(self.df.loc[i, source]) > 0:
437                         values = [a['text_blob']['polarity'] for a in
    self.df.loc[i, source]]
438                         score = (self.ss_decay * previous) + sum(values
    )
439                     else:
440                         score = (self.ss_decay * previous)
441                     scores.append(score)
442                     previous = score
443                 self.df[f'tb_{source}_score'] = scores
444 
445             if vader:
446                 if verbose: print("Vader")
447                 analyzer = SentimentIntensityAnalyzer()
448                 for i in range(len(self.df)):
449                     for a in self.df.loc[i, source]:
450                         a['vader'] = analyzer.polarity_scores(a['title'
    ])
451 
452                 if verbose: print("Calculating Vader scores")
453                 previous = 0
454                 scores = []
455                 for i in range(len(self.df)):
456                     if len(self.df.loc[i, source]) > 0:
457                         values = [a['vader']['compound'] for a in self.
    df.loc[i, source]]
```

```
458                         score = (self.ss_decay * previous) + sum(values
    )
459                     else:
460                         score = (self.ss_decay * previous)
461                     scores.append(score)
462                     previous = score
463                 self.df[f'vader_{source}_score'] = scores
464 
465     def calculate_cheat_values(self):
466         cheats = []
467         for i in range(0, len(self.df) - 1):
468             cheats.append((self.df.loc[i+1,'close'] / self.df.loc[i,'
    close']) - 1)
469         cheats.append(0)
470         self.df['cheats'] = cheats
471 
472     # Compound Methods
473     def extract_and_calculate_basic(self, verbose=True):
474         if verbose: print("Extracting investment ranking data")
475         self.extract_investment_ranking_data()
476         if verbose: print("Calculating technical indicators")
477         self.calculate_technical_indicators()
478         if verbose: print("Calculating cheat values")
479         self.calculate_cheat_values()
480 
481     def extract_and_calculate_all(self, verbose=True):
482         self.extract_and_calculate_basic(verbose=verbose)
483         if verbose: print("Extracting news data")
484         self.extract_news_data(investing=False)
485         if verbose: print("Calculating news sentiment")
486         self.calculate_news_sentiment()
487 
488     # Other Methods
489     def save_as_excel(self):
490         self.df.to_excel(f"{self.code}.xlsx")
491 
492     @staticmethod
493     def _iso_to_datetime(string_date):
494         try:
495             return datetime(*[int(d) for d in string_date.split("-")])
496         except ValueError as e:
497             print(f"Failed to convert '{string_date}'")
498 
499     @staticmethod
500     def get_google_news_driver(headless=False):
501         if WEBDRIVER_PATH == "other/chromedriver/":
502             raise Exception("No webdriver found, add webdriver file to
    other/chromedriver folder")
503         # Setup webdriver
504         options = webdriver.ChromeOptions()
505         if headless: options.add_argument('headless')
506         options.add_argument('window-size=1200x600')
507         driver = webdriver.Chrome(WEBDRIVER_PATH, chrome_options=
    options)
508         # Accept conditions
509         driver.get("https://www.google.co.uk/search?")
510         button_xpath = '//*[@id="L2AGLb"]'
```

```
511         WebDriverWait(driver, 5).until(EC.presence_of_element_located((
    By.XPATH, button_xpath)))
512         button = driver.find_element_by_xpath(button_xpath)
513         button.click()
514         url = ("https://www.google.co.uk/search?q=Apple&"
515             "tbs=cdr:1,cd_min:01/01/2014,cd_max:12/31/2014&tbm=nws")
516         driver.get(url)
517         try:
518             driver.find_element_by_xpath('//*[@id="captcha-form"]')
519             time.sleep(2)
520             print("\nThere appears to be Captcha form. Complete this
    and then press enter.")
521             input()
522         except selenium.common.exceptions.NoSuchElementException as e:
523             pass
524         return driver
```

B.3 Storage

These functions are used to store and retrieve Stock objects which may contain stock data. A function is also included to extract all stock objects from a given folder.

```
1  import dill
2  from datetime import datetime
3  from os import listdir
4
5
6  def save_dill_object(my_object, file_name):
7      with open(file_name, "wb") as file:
8          dill.dump(my_object, file)
9
10 def retrieve_dill_object(file_name):
11     with open(file_name, "rb") as file:
12         my_object = dill.load(file)
13     return my_object
14
15 def save_stock(stock, folder):
16     code = stock.code
17     date = datetime.today().strftime('%Y-%m-%d')
18     save_dill_object(stock, f"{folder}/{code}_{date}.dill")
19
20 def retrieve_stocks_from_folder(folder):
21     stocks = []
22     files = listdir(folder)
23     for stock_file in files:
24         s = retrieve_dill_object(f"{folder}/{stock_file}")
25         stocks.append(s)
26     return stocks
```

B.4 Constants

This file contains the constants used in the software. For instance, it includes a dictionary for converting investment ratings to values and a list of the top 100 S&P 500 stocks.

```
import os


TRADING_DAYS = 252
BASE_RATE = 0.02

# Data Scraping

MONTHS = {'jan': "01", 'feb': "02", 'mar': "03", 'apr': "04", 'may': "05", 'jun': "06", 'jul': "07", 'aug': "08", 'sep': "09", 'oct': "10", 'nov': "11", 'dec': "12"}

STANDARD_HEADERS = {'User-Agent':('Mozilla/5.0 (Windows NT 6.1; WOW64)' ' AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2227.0 Safari/537.36')}

folder_contents = os.listdir("other/chromedriver")
files = [file for file in folder_contents if file.startswith("chromedriver")]
file_name = files[0] if len(files) > 0 else ""
WEBDRIVER_PATH = f"other/chromedriver/{file_name}"

# Sentiment Analysis

RANKING_VALUES = {
    "Cautious": -1,
    "Conviction Buy": 1,
    "Below Average": -1,
    "Above Average": 1,
    "Underperformer": -1,
    "Overperformer": 1,
    "Fair Value": 0,
    "": 0,
    "Mixed": 0,
    "Reduce": -1,
    "Hold Neutral": 0,
    "Long-Term Buy": 1,
    "Equal-Weight": 0,
    "Buy": 1,
    "Strong Buy": 1,
    "Hold": 0,
    "In-Line": 0,
    "Perform": 0,
    "Negative": -1,
    "Positive": 1,
    "Overweight": 1,
    "Sector Outperform": 1,
    "Top Pick": 1,
    "Accumulate": 1,
    "Market Perform": 0,
    "Market Outperform": 1,
    "Sector Perform": 0,
    "In-line": 0,
    "Equal-weight": 0,
    "Neutral": 0,
    "Average": 0,
    "Peer Perform": 1,
    "Underperform": -1,
```

```
    "Market Underperform": -1,
    "Sector Weight": 0,
    "Sector Underperform": -1,
    "Market Weight": 0,
    "Long-term Buy": 1,
    "Underweight": -1,
    "Sell": -1,
    "Outperform": 1
}

CHANGE_VALUES = {
    'init': 0,
    'reit': 0,
    'down': -1,
    'up': 1,
    'main': 0
}

HF_LABEL_VALUES = {
    'NEGATIVE' : -1,
    'POSITIVE' : 1
}

# Stock Arguments

SNP_500_TOP_100 = [
    ['Apple', 'AAPL', 'apple-computer-inc'],
    ['Microsoft', 'MSFT', 'microsoft-corp'],
    ['Amazon', 'AMZN', 'amazon-com-inc'],
    ['Alphabet', 'GOOG', 'google-inc-c'],
    ['Facebook', 'FB', 'facebook-inc'],
    ['Berkshire Hathaway', 'BRK-B', 'berkshire-hathaway'],
    ['Tesla Motors', 'TSLA', 'tesla-motors'],
    ['Visa', 'V', 'visa-inc'],
    ['Nvidia', 'NVDA', 'nvidia-corp'],
    ['J P Morgan', 'JPM', 'jp-morgan-chase'],
    ['Johnson and Johnson', 'JNJ', 'johnson-johnson'],
    ['Wal-Mart', 'WMT', 'wal-mart-stores'],
    ['United Health', 'UNH', 'united-health-group'],
    ['Mastercard', 'MA', 'mastercard-cl-a'],
    ['Bank of America', 'BAC', 'bank-of-america'],
    ['Procter and Gamble', 'PG', 'procter-gamble'],
    ['Home Depot', 'HD', 'home-depot'],
    ['Walt Disney', 'DIS', 'disney'],
    ['Adobe', 'ADBE', 'adobe-sys-inc'],
    ['Comcast', 'CMCSA', 'comcast-corp-new'],
    ['Exxon Mobil', 'XOM', 'exxon-mobil'],
    ['Coca-Cola', 'KO', 'coca-cola-co'],
    ['Verizon Communications', 'VZ', 'verizon-communications'],
    ['Salesforce', 'CRM', 'salesforce-com'],
    ['Intel', 'INTC', 'intel-corp'],
    ['Netflix', 'NFLX', '"netflix,-inc."'],
    ['Oracle', 'ORCL', 'oracle-corp'],
    ['Cisco', 'CSCO', 'cisco-sys-inc'],
    ['Pfizer', 'PFE', 'pfizer'],
    ['Eli Lilly', 'LLY', 'eli-lilly-and-co'],
    ['AT&T', 'T', 'at-t'],
    ['NIKE', 'NKE', 'nike'],
```

```
113     ['PepsiCo', 'PEP', 'pepsico'],
114     ['AbbVie', 'ABBV', 'abbvie-inc'],
115     ['Chevron', 'CVX', 'chevron'],
116     ['Abbott Labs', 'ABT', 'abbott-laboratories'],
117     ['Merck', 'MRK', 'merck---co'],
118     ['Broadcom', 'AVGO', 'avago-technologies'],
119     ['Thermo Fisher Scientific', 'TMO', 'thermo-fisher-sc'],
120     ['Danaher', 'DHR', 'danaher-corp'],
121     ['T-Mobile US', 'TMUS', 'metropcs-communications'],
122     ['Accenture', 'ACN', 'accenture-ltd'],
123     ['Wells Fargo', 'WFC', 'wells-fargo'],
124     ['United Parcel Service', 'UPS', 'united-parcel'],
125     ["McDonald's", 'MCD', 'mcdonalds'],
126     ['Texas Instruments', 'TXN', 'texas-instru'],
127     ['Costco', 'COST', 'costco-whsl-corp-new'],
128     ['Medtronic', 'MDT', 'medtronic'],
129     ['Morgan Stanley', 'MS', 'morgan-stanley'],
130     ['Philip Morris International', 'PM', 'philip-morris-intl'],
131     ['Qualcomm', 'QCOM', 'qualcomm-inc'],
132     ['Bristol-Myers Squibb', 'BMY', 'bristol-myer-squiib'],
133     ['Honeywell International', 'HON', 'honeywell-intl'],
134     ['Linde', 'LIN', 'linde-plc'],
135     ['NextEra Energy', 'NEE', 'nextera-energy-inc'],
136     ['Union Pacific', 'UNP', 'union-pacific'],
137     ['Citigroup', 'C', 'citigroup'],
138     ['Boeing', 'BA', 'boeing-co'],
139     ['Amgen', 'AMGN', 'amgen-inc'],
140     ['Lowes Co.', 'LOW', 'lowes-companies'],
141     ['Charles Schwab', 'SCHW', 'charles-schwab'],
142     ['Raytheon Technologies', 'RTX', 'united-tech'],
143     ['Intuit', 'INTU', 'intuit'],
144     ['Charter Communications', 'CHTR', 'charter-communications'],
145     ['Starbucks', 'SBUX', 'starbucks-corp'],
146     ['BlackRock', 'BLK', '"blackrock,-inc.-c"'],
147     ['IBM', 'IBM', 'ibm'],
148     ['American Express', 'AXP', 'american-express'],
149     ['American Tower REIT', 'AMT', 'amer-tower-corp'],
150     ['Applied Materials', 'AMAT', 'applied-matls-inc'],
151     ['Goldman Sachs', 'GS', 'goldman-sachs-group'],
152     ['Caterpillar', 'CAT', 'caterpillar'],
153     ['Target', 'TGT', 'target'],
154     ['General Electric', 'GE', 'general-electric'],
155     ['3M', 'MMM', '3m-co'],
156     ['CVS Health', 'CVS', 'cvs-corp'],
157     ['Estee Lauder Companies', 'EL', 'estee-lauder'],
158     ['Lockheed Martin', 'LMT', 'lockheed-martin'],
159     ['ServiceNow', 'NOW', 'servicenow-inc'],
160     ['Intuitive Surgical', 'ISRG', 'intuitive-surgical-inc'],
161     ['Advanced Micro Devices', 'AMD', 'adv-micro-device'],
162     ['Deere and Co.', 'DE', 'deere---co'],
163     ['Stryker', 'SYK', 'stryker'],
164     ['S&P Global', 'SPGI', 'mcgraw-hill'],
165     ['Booking Holdings', 'BKNG', 'priceline.com-inc'],
166     ['Anthem', 'ANTM', 'wellpoint-inc'],
167     ['Fidelity National Information', 'FIS', 'fidelity-natl-in'],
168     ['Prologis', 'PLD', 'prologis'],
169     ['Zoetis', 'ZTS', 'zoetis-inc'],
170     ['Lam Research', 'LRCX', 'lam-research-corp'],
```

```
    ['Mondelez', 'MDLZ', 'mondelez-international-inc'],
    ['Micron', 'MU', 'micron-tech'],
    ['Altria', 'MO', 'altria-group'],
    ['US Bancorp', 'USB', 'us-bancorp'],
    ['General Motors', 'GM', 'gen-motors'],
    ['Crown Castle Intl', 'CCI', 'crown-castle-int'],
    ['Gilead Sciences', 'GILD', 'gilead-sciences-inc'],
    ['Automatic Data Processing', 'ADP', 'auto-data-process']
]
```

B.5 Plotting

This function was used during the experimentation to produce graph images and statistics on the performance of agents.

```
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use('Agg')

def plots_and_stats(experiment_name, name_of_independent, folder,
    log_scale=True, include_sr=True):
    training_dict = {}
    testing_dict = {}
    for file in listdir(folder):
        if file == ".gitignore":
            continue
        file_name = file[:-5]
        components = file_name.split("_")
        period = components[-1]
        name = "_".join(components[:-1])
        file_path = f"{folder}/{file}"
        df = pd.read_excel(file_path, index_col=0)
        if period == "training":
            training_dict[name] = df
        elif period == "testing":
            testing_dict[name] = df
        else:
            raise ValueError("Neither training or testing")

    # Plot training
    for name in training_dict:
        # Calculate Averages
        df = training_dict[name]
        averages = []
        for y in range(len(df)):
            values = [df[x][y] for x in range(1,len(df.columns))]
            returns = [(v / 1_000_000) - 1 for v in values]
            average = sum(returns) / len(returns)
            averages.append(average)
        # Plot Averages
        plt.plot(averages, label=name)
    title = f"{experiment_name} Training"
    # plt.title(title)
    plt.xlabel("Episodes")
    plt.ylabel("Average Return")
    if log_scale: plt.yscale('log')
```

```
    plt.legend()
    plt.tight_layout()
    plt.savefig(f"data/plots/{title}")
    plt.clf()

    # Plot testing
    if include_sr:
        testing_stats = pd.DataFrame({"Agent": ["Average Return", "Std Devs", "Sample Size", "SM Std Devs", "Average Sharpe Ratio", "Sharpe Ration SM std Devs"]})
    else: testing_stats = pd.DataFrame({"Agent": ["Average Return", "Std Devs", "Sample Size", "SM Std Devs"]})
    average_final_values = []
    errs = []
    average_sharpe_ratio_values = []
    sr_errs = []
    names = []
    for name in testing_dict:
        df = testing_dict[name]
        values = [df[x][0] for x in range(1,len(df.columns))]
        returns = [(v / 1_000_000) - 1 for v in values]
        average = sum(returns) / len(returns)
        average_final_values.append(average)
        std_dev = np.std(returns)
        err = std_dev / math.sqrt(len(returns))
        errs.append(err)

        if include_sr:
            sharpe_ratios = [df[x][2] for x in range(1,len(df.columns))]
            average_sharpe_ratio = sum(sharpe_ratios) / len(sharpe_ratios)
            average_sharpe_ratio_values.append(average_sharpe_ratio)
            sr_err = np.std(sharpe_ratios) / math.sqrt(len(sharpe_ratios))
            sr_errs.append(sr_err)

        names.append(name)

        if include_sr:
            testing_stats[name] = [average, std_dev, len(returns), err, average_sharpe_ratio, sr_err]
        else: testing_stats[name] = [average, std_dev, len(returns), err]
    title = f"{experiment_name} Testing"
    # plt.title(title)
    if log_scale: plt.xscale('log')
    plt.barh(names, average_final_values, xerr=errs, ec="black", capsize=5)
    plt.xlabel("Average Return")
    plt.ylabel(name_of_independent)
    plt.tight_layout()
    plt.savefig(f"data/plots/{title}")
    plt.clf()
    if include_sr:
        title = f"{experiment_name} Testing Sharpe Ratios"
        # plt.title(title)
        if log_scale: plt.xscale('log')
```

```
90          plt.barh(names, average_sharpe_ratio_values, xerr=sr_errs, ec="
    black", capsize=5)
91          plt.xlabel("Average Sharpe Ratio")
92          plt.ylabel(name_of_independent)
93          plt.tight_layout()
94          plt.savefig(f"data/plots/{title}")
95          plt.clf()
96          testing_stats.to_excel(f"data/testing_stats/{experiment_name}.
    xlsx")
```

Appendix C

Ethics Checklist

Each page of the department of computer science 12-point ethics checklist for this project has been included in Figures C.1, C.2 and C.3.

This form must be attached to the dissertation as an appendix.

UNIVERSITY OF BATH

Department of Computer Science
12-Point Ethics Checklist for UG and MSc Projects

Student Oscar Wignall (ow294)

Academic Year or Project Title 2020/2021

Supervisor James Laird & George Fletcher

Does your project involve people for the collection of data other than you and your supervisor(s)? ~~YES~~ / (NO)

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Will you prepare a Participant Information Sheet for volunteers?* YES / NO
   This means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO
   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Will there be any intentional deception of the participants?* YES / NO
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO
   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature

Figure C.1: Ethics checklist page 1

of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. ***Will participants voluntarily give informed consent?*** YES / NO
Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete. If your data collection is entirely anonymous and does not include collection of personal data you do not need to collect a signature. Instead, you should include a checkbox, which must be checked by the participant to indicate that informed consent has been given.

6. ***Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?*** YES / NO
Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. ***Will you be offering any incentive to the participants?*** YES / NO
The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. ***Will you be in a position of authority or influence over any of your participants?*** YES / NO
A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. ***Will any of your participants be under the age of 16?*** YES / NO
Parental consent is required for participants under the age of 16.

10. ***Will any of your participants have an impairment that will limit Their understanding or communication?*** YES / NO
Additional consent is required for participants with impairments.

11. ***Will the participants be informed of your contact details?*** YES / NO
All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

Figure C.2: Ethics checklist page 2

12. ***Will you have a data management plan for all recorded data?*** YES / NO
Personal data is anything which could be used to identify a person, or which can be related to an identifiable person. All personal data (hard copy and/or soft copy) should be anonymized (with the exception of consent forms) and stored securely on university servers (not the cloud).

Figure C.3: Ethics checklist page 3