Tinbergen Institute

Business Data Science

March 3, 2023

Stock trading using Reinforcement Learning.

Individual Assignment.

Course:
Reinforcement Learning

Teachers:
prof. dr. Gui Liberali

Author:
Dewy Jungslager

Abstract

This paper shows that the use of implied volatility, on-balance volume, relative strength index, and previous day returns as contextual variables can be valuable for predicting individual stock performance. These variables helped the model avoid substantial drawdowns, resulting in large outperformance compared to other methods. Additionally, I show an approach to portfolio management based on random simulations. This remedies the problem of choosing portfolio weights, as the model decides which portfolio weights are best. I show that superior results can be obtained using random portfolios compared to individual stocks for the vanilla Upper Confidence Bound and Thompson Sampling algorithms.

Keywords: Reinforcement Learning, Stock Trading, Portfolio Management, Upper Confidence Bound, Thompson Sampling, Portfolio Weight Simulation.

1 Introduction

The use of machine learning algorithms has gained popularity in recent years as a means of predicting the future prices of stocks in the stock market. Reinforcement learning (RL), a subfield of machine learning, has emerged as a popular approach for making trading decisions in financial markets. A comprehensive survey was written by Sun et al. (2021) about RL in quantitative trading. RL involves training an agent to learn from past experiences and adjust its decision-making strategy accordingly. RL-based trading algorithms have shown promising results in various financial applications, including stock trading.

In stock trading, the goal is to maximize returns by making informed buy and sell decisions. A common approach is to use bandit algorithms, which select actions based on past rewards. The traditional bandit approach assumes that rewards are independent of the context or state of the environment and only considers the current action. However, in financial markets, rewards are often dependent on the current market conditions and the history of the market. To capture this dependence, contextual bandits have been introduced, which take into account the current state of the environment as well as the past rewards when selecting actions.

Many papers in the reinforcement literature, use very complex methods such as deep neural networks in combination with reinforcement learning (see for example Liu et al., 2018, Wu et al., 2020, Yang et al., 2020) for stock market predictions. However, these models require a lot of data and overfit easily. One of the problems in stock market data is that models trained on past data suffer from historical bias. Therefore, simpler algorithms such as the Upper Confidence Bound (UCB) and Thompson Sampler (TS) might be more practical. These methods are easy to implement and require little computational power. This makes it attractive for practitioners.

Contextual bandits have outperformed traditional bandits in various applications, such as in online advertising where they have been used to personalize ad recommendations based on user context, resulting in improved click-through rates and revenue. In the context of stock trading, contextual bandits can be used to incorporate information such as option data and traditional indicators like volume and relative strength index. Option data provides valuable information about future stock price expectations, while traditional indicators are based on past market behavior and provide insight into the current market conditions.

By incorporating these contextual features into the bandit algorithm, we can potentially improve the trading strategy and achieve better returns. The use of contextual bandits offers a promising approach for selecting stocks to trade in the stock market. Moreover, the contextual features can also be extended to the portfolio level, where instead of choosing individual stocks, the bandit can select a portfolio of stocks. This approach to portfolio construction involves randomly simulating multiple portfolio weights and allowing the bandit to choose the best portfolio weights. This comes with the advantage that we do not have to choose portfolio weights or model them.

The research question that arises is whether it is beneficial to use contextual bandits for portfolio construction instead of individual stock selection. Can the contextual information provide a better understanding of the market and lead to more profitable investments? To answer this question, this study aims to compare the performance of the contextual bandit algorithm in both scenarios. The results of this study can provide insights into the effectiveness of contextual bandits in portfolio construction and can be of value to investors and traders in the stock market.

Overall, this study aims to investigate the potential benefits of using contextual bandits in stock trading and portfolio construction. By incorporating contextual information into the bandit algorithm, we can potentially improve the decision-making process and achieve better returns. The results of this study can provide valuable insights into the effectiveness of RL-based trading strategies in financial markets and can contribute to the development of more effective trading algorithms.

The rest of this paper is structured as follows. Section 2 discusses the algorithms, which contextual variables are used, and how the portfolios are constructed. Then in Section 3 I discuss the data. I continue with the empirical application in Section 4. Here I look at the sensitivity of the algorithms and compare the different methods to eachother. Section 5 discusses some of the limitations of the methods and explores directions of futher research based on these limitations. Finally, I conclude in Section 6.

2 Methodology

In this section I introduce the methodology that is applied in this paper. Additionally I will discuss some of the advantages and disadvantages of the different methods.

2.1 The Upper Condidence Bound

The Upper Confidence Bound (UCB) algorithm is a widely used reinforcement learning technique that can be used to solve the multi-armed bandit problem. The basic idea behind the UCB algorithm is to balance exploration and exploitation by maintaining estimates of the expected rewards of each action and selecting the action with the highest upper confidence bound.

In the context of stock trading, the UCB algorithm can be used to select which stocks to invest in based on their expected returns. The algorithm maintains estimates of the expected returns of each stock and selects the stock with the highest upper confidence bound.

Advantages of the UCB algorithm include its simplicity and ease of implementation. It is also easy to interpret and understand, which makes it popular among practitioners. Another advantage is that the algorithm can be used in a wide range of applications, including stock trading, website optimization, and online advertising.

One disadvantage of the UCB algorithm is that it can be sensitive to the choice of hyperparameters, particularly the exploration parameter. If the exploration parameter is set too high, the algorithm may explore too much and miss out on potential gains. If the exploration parameter is set too low, the algorithm may not explore enough and get stuck in a suboptimal solution.

Another disadvantage of the UCB algorithm is that it assumes that the rewards of each action are independent and identically distributed. This assumption may not hold in practice, particularly in the context of stock trading where the rewards of different stocks may be correlated.

Despite its limitations, the UCB algorithm can be useful for stock trading decisions because it provides a systematic way to balance exploration and exploitation. By exploring different stocks and updating estimates of their expected returns, the algorithm can identify promising stocks to invest in and avoid investing in underperforming stocks.

The LinUCB algorithm is an extension of the UCB algorithm that incorporates contextual information into the decision-making process. It is particularly useful in scenarios where there are multiple options (e.g., stocks) and multiple features that may impact the outcome (e.g., historical stock prices, market trends, option data, etc.).

LinUCB is based on ridge regression, which is a regularization technique that adds a penalty term to the least squares optimization problem. This penalty term helps prevent overfitting by shrinking the coefficients towards zero. In the case of LinUCB, the regularization term is used to prevent the algorithm from placing too much emphasis on any particular feature.

The LinUCB algorithm operates by first learning a linear model that relates the features to the rewards. This model is updated at each time step as new data becomes available. The algorithm then selects the arm with the highest estimated reward based on the current feature vector.

One advantage of LinUCB over other contextual bandit algorithms is its ability to efficiently and effectively handle high-dimensional feature spaces. It achieves this by using a linear combination of feature vectors from previous rounds to approximate the feature vector of the current round. This reduces the number of parameters that need to be learned and allows the algorithm to make more accurate predictions with fewer data points.

Another advantage of LinUCB is that it is computationally stable and can utilize existing implementations of ridge regression. This makes it easier to implement and reduces the risk of numerical instability that can occur with other methods.

However, like any algorithm, LinUCB has its limitations and assumptions. One assumption is that the reward function is linearly related to the features. This may not always be the case in real-world scenarios, where non-linear relationships may exist. Another limitation is that the algorithm assumes that the features are stationary over time. If the features change significantly over time, the algorithm may not be able to make accurate predictions.

In terms of stock trading decisions, the LinUCB algorithm can be useful in identifying the best stocks to invest in based on a variety of factors. For example, it can be used to analyze historical stock prices, market trends and option data to make predictions about future stock prices. By incorporating contextual information into the decision-making process, the algorithm can help investors make more informed decisions and achieve better returns.

2.2 Thompson Sampling

Thompson Sampling (TS) is a another popular probabilistic algorithm for solving the exploration-exploitation trade-off problem in multi-armed bandit settings. The basic idea of TS is to model the underlying distribution of each arm's reward and then sample from those distributions to select which arm to play in each round. Specifically, in each round, TS selects an arm according to the probability that it is the optimal arm, given the observed data up to that point. This probability is computed by drawing a sample from the posterior distribution over the arm's reward distribution, which is updated after each round based on the observed rewards.

One advantage of TS over other methods such as UCB is that it is based on randomization, which can inject noise into the algorithm and prevent it from getting stuck in suboptimal solutions. This can lead to better performance in certain scenarios, particularly when the underlying reward distributions are complex or non-stationary. However, this randomization can also introduce additional variance in the algorithm, which can lead to inferior performance in certain scenarios.

Contextual Thompson Sampling (CTS) is an extension of TS that incorporates contextual information into the algorithm. In CTS, each arm is associated with a feature vector, and the algorithm models the reward distribution for each arm as a function of the features. Specifically, CTS assumes that the reward for arm i in round t follows a Gaussian distribution with mean $\mu_t(i)$ and variance $\sigma_t^2(i)$, where $\mu_t(i)$ is a linear function of the features $x_t(i)$ for arm i in round t. The algorithm then uses the observed rewards and features to update the posterior distribution over the model parameters, which are then used to compute the probability of each arm being optimal in the current round.

One advantage of CTS over non-contextual methods like TS and UCB is that it can take into account the effect of contextual features on the reward distribution, allowing for more accurate and personalized recommendations. This can be particularly useful in stock trading and portfolio selection, where the performance of different assets may depend on factors such as industry, sector, and macroeconomic conditions. However, CTS also requires more computational resources and may be more sensitive to overfitting than non-contextual methods.

In summary, both UCB and TS are effective methods for solving the exploration-exploitation trade-off problem in multi-armed bandit settings, with different trade-offs in terms of performance and computational complexity. CTS is an extension of TS that can incorporate contextual information into the algorithm, providing more personalized recommendations in certain scenarios, but at the cost of increased computational complexity and potential overfitting. The assumptions made by each method should be carefully considered in the context of the problem at hand, particularly in stock trading and portfolio selection where the underlying reward distributions may be complex and non-stationary.

2.3 Contextual Variables

The purpose of this section is to introduce the variables that are used as contextual variables. I will highlight how they are obtained and what kind of information they contain.

Implied volatility based on American call options is a measure of the market's expectation for future volatility in the underlying asset, i.e. the stock. It is obtained by solving the Black-Scholes equation for the volatility parameter, given the market price of the call option, the strike price, the time to expiration, the current stock price, and the risk-free interest rate. Implied volatility can be interpreted as a measure of the uncertainty or risk associated with the stock price, as perceived by the market. By averaging the implied volatility over call options with different expiration dates, you are able to incorporate forward-looking data into your model while mitigating the effects of missing data for daily trading.

On-balance volume (OBV) is a technical analysis indicator that measures the buying and selling pressure on a stock. It is obtained by summing the volume of shares traded on days when the stock price closes higher than the previous day's closing price and subtracting the volume of shares traded on days when the stock price closes lower. OBV can be interpreted as an indicator of the market sentiment towards the stock - if more shares are being bought than sold, it suggests bullish sentiment, while if more shares are being sold than bought, it suggests bearish sentiment.

Relative strength index (RSI) is another technical analysis indicator that measures the strength of a stock's price movement. It is obtained by calculating the average gain and loss of the stock over a given period of time (typically 14 days), and then calculating the ratio of the average gain to the average loss. RSI can be interpreted as an indicator of whether a stock is overbought or oversold - if the RSI is above 70, it suggests that the stock is overbought and may be due for a correction, while if the RSI is below 30, it suggests that the stock is oversold and may be due for a rebound.

Finally, the previous log returns are obtained simply by calculating the natural logarithm of the ratio of the current closing price to the previous closing price. This provides a measure of the stock's historical performance, which can be useful in identifying trends or patterns in the stock's price movement.

These variables are useful as contextual variables in the algorithms to select stocks because they provide a range of information about the stock, including its perceived risk, market sentiment, price momentum, and historical performance. By incorporating these variables into the model, you are able to capture a more complete picture of the stock's underlying dynamics and make more informed trading decisions. Additionally, these variables have been widely studied and shown to be effective in predicting stock prices and identifying profitable trading strategies, making them a valuable tool for stock traders and investors.

2.4 Portfolio construction

Portfolio construction is a critical aspect of trading and investment, as it involves choosing the appropriate weights of assets to maximize returns while minimizing risks. However, determining the optimal weights of assets in a portfolio can be challenging, as it requires balancing several factors, including individual asset performance, risk appetite, and market conditions. In this study, we propose a methodology to construct portfolios using contextual bandits that leverage contextual variables such as implied volatility, relative strength index, and previous log returns.

To construct our portfolios, we first randomly simulate portfolio weights from a normal distribution. This approach yields multiple portfolios with varying weights of assets, which enables us to explore a range of portfolio compositions. I then construct the returns and rewards for each portfolio. The contextual variables used in the portfolio construction include the relative strength index, previous log returns, and the implied volatility calculated as a weighted average of the implied volatility on the individual stocks based on the corresponding portfolio weights.

Notably, while the OBV was a useful contextual variable for selecting individual stocks, it does not generalize to portfolios in a useful manner.

The proposed approach has several advantages over traditional portfolio construction methods. First, it enables us to incorporate forward-looking data from options markets, which can provide valuable information about future volatility and potential returns. Second, it leverages a range of contextual variables that capture various aspects of market conditions, including historical performance and current momentum. Finally, by randomly simulating portfolio weights, we can explore a range of portfolio compositions and avoid potential biases that may arise from using predetermined weights.

In summary, our proposed methodology for constructing portfolios using contextual bandits and random portfolio weight simulations is a promising approach for traders and investors looking to optimize their portfolios. By incorporating a range of contextual variables and exploring a range of portfolio compositions, we can identify optimal weights of assets that maximize returns while minimizing risks.

3 Data

For this study, 17 assets were randomly selected from the S&P 500 index. Daily closing prices adjusted for stock splits and dividends were obtained from Yahoo Finance for the period spanning from January 23rd, 1998, to December 31st, 2021. The corresponding daily volumes for these assets were also obtained from Yahoo Finance.

The period from 1998 to 2021 was a particularly interesting time for stock trading and portfolio management, marked by several significant financial crises and events. The dot-com bubble in the late 1990s saw a surge in stock prices for many internet-based companies, followed by a sharp correction in 2000-2002. The Great Financial Crisis of 2008, caused by the collapse of the housing market and subprime mortgages, resulted in a severe economic recession and a significant drop in stock prices. The European Sovereign Debt Crisis in the early 2010s, triggered by high levels of government debt and weak economic growth, further added to the turmoil in global financial markets.

These events have made it a challenging time for investors and fund managers to navigate the markets and construct profitable portfolios. The use of quantitative methods and algorithmic trading have become increasingly popular during this period, as investors seek to extract signals from large volumes of data and mitigate risks.

To analyze the performance of these assets, daily log returns were calculated using the closing prices. In addition, daily American call option prices were obtained from OptionMetrics. From these option prices, daily implied volatilities were derived using the Black-Scholes model. The option price range is the same as for the daily returns.

To address the issue of missing values in the option data, the daily implied volatilities were obtained as an equally weighted average of the available daily implied volatilities with expiration date 7 to 30 days (approximately one week to a month) into the future. If there were still missing values, they were imputed using a linear interpolation method.

After using starting observations for the contextual variables, I am left with 6009 observations per asset.

4 Empirical Application

For the empirical application I consider the performance of the methods by trading on real stock data. Here we have to be careful to only use the information that is available at each time point, to not give the model an unfair advantage. The reward at time t is constructed as the return of buying the asset at time t and selling it at time $t + 1$.

In this empirical application I apply the UCB, contextual UCB (CUCB), Thompson sampling, and contextual Thompson Sampling (CTS) algorithms. For the contextual algorithms I use the contextual variables discussed in 2.3. At time t the available information is the on-balance volume at time t, the implied volatility at time t, the relative strength index computed over the last 15 trading days (3 weeks of trading), and the return from time $t-1$ to t.

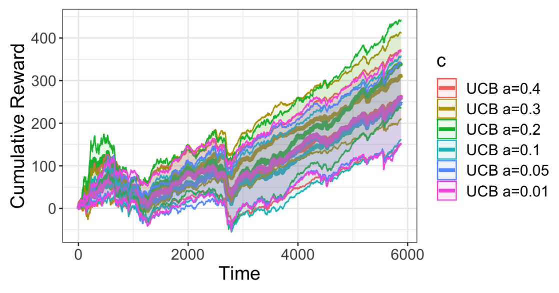
The first question that I investigate in this experiment is if we can improve upon the vanilla UCB and TS algorithms using the contextual variables discussed above. The second question I answer is whether it also holds at the portfolio level. Finally, I compare the performance of individual stock trading against portfolio investing. Individual stock trading has the advantage of easily taking individual stock information such as on-balance volume into account, while it is more difficult to aggregate this to the portfolio level. The potential benefit of portfolio investing is that you are diversified over all available assets. However, this can also come with the downside that you are invested in the losing stocks at all times and could have given more weight to the winning stocks.

I also perform a sensitivity analysis where I look at the effect of the hyperparameters of the UCB and TS algorithms, and the effect of different amount of portfolios. For all simulations I use a simulation size of $N = 100000$ and number of simulation $m = 14$.

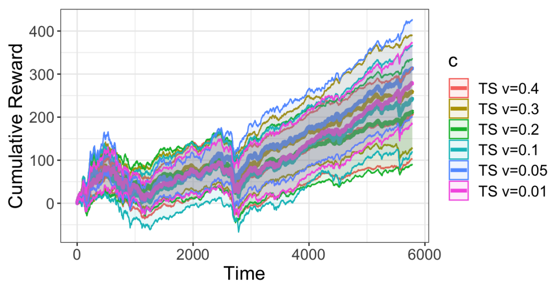
4.1 Hyperparameter Study

Both the UCB and TS algorithm are based on a hyperparameter that determines the trade-off between exploration and exploitation. I start by roughly investigating which parameter settings are optimal for this application.

Figure 1 shows the performance of the UCB (panel a) and TS (panel b) algorithms for individual stock picking. The plots shows how the algorithm performs for different values of the hyperparameters. For both the algorithms, the larger the hyperparameter the more the algorithm gives weight to exploration compared to exploitation.



(a)

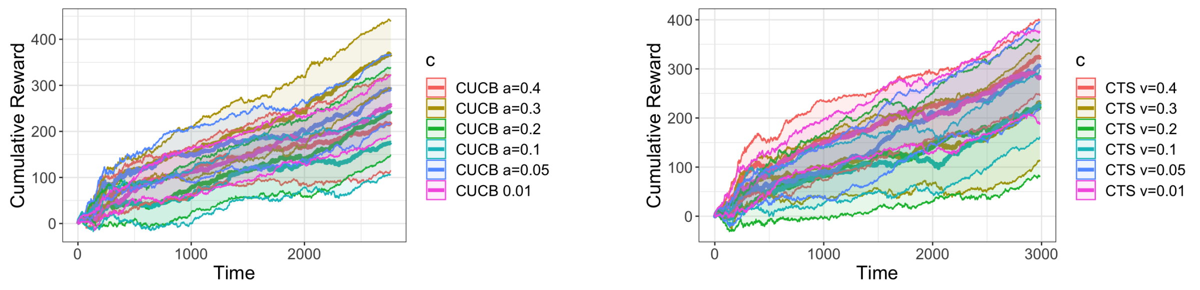


(b)

Figure 1: This plot shows the performance of the UCB algorithm (panel a) and the TS algorithm (panel b) for different values of the hyperparameters for individual stock picking. Simulation size $N = 100000$ *and number of simulation* $M = 14$.

In the figure we can see that there is a substantial difference between the performance of the algorithms for the different values of the hyperparameters. The UCB algorithm performs best for $\alpha = 0.2$. In close second we can see that the algorithm also performs well for $\alpha = 0.3$. For all the other values considered the performance is very similar and considerably worse. The TS algorithm performs best for $\nu = 0.05$. All the other values of ν perform quite different from each other. Interestingly, for both the UCB and TS algorithm there is a non-linear relation between the performance of the algorithm and the value of the hyperparameter.

Figure 2 shows the performance of the CUCB (panel a) and the CTS (panel b) algorithms for individual stock picking. Again, the performance is shown for different values of the hyperparameters.



(a) (b)

Figure 2: This plot shows the performance of the CUCB algorithm (panel a) and the CTS algorithm (panel b) for different values of the hyperparameters for individual stock picking. Simulation size $N = 100000$ *and number of simulation* $M = 14$.

For the CUCB algorithm the best performance is obtained for $\alpha = 0.3$. Also for the vanilla UCB algorithm this value performed well. Note, due to the use of contextual variables we end up with less observations in the simulation. In Figure 1 there are approximately 6000 time observations, while in Figure 2 there are approximately 3000 time observations. After around 3000 observations also for the vanilla UCB $\alpha = 0.3$ performs best. Therefore, for the final comparison between the methods I choose $\alpha = 0.3$.

The CTS algorithm performs best for $\nu = 0.4$. Interestingly, this value was one of the worst performing values for the vanilla TS algorithm. This indicates that the CTS algorithm favors much more exploration compared to the vanilla TS algorithm.

Remarkably, The vanilla UCB and TS algorithms experience substantial draw-downs from $t = 600$ to $t = 1200$ and again from $t = 2400$ to $t = 2800$. Such draw-down periods almost do not occur for the contextual algorithms. These drops can occur due to changing financial conditions, such as crisis. As discussed in Section 3, there are periods of substantial financial turmoil in the data, that could explain these drops. The contextual variables might be able to capture these changing financial conditions and adapt the strategy based on this. The contextual variables are chosen to capture trends in the stock price (such as relative strength index and on-balance volume) and future expectations (via forward looking option data). The trends can indicate to the model when an up-trend changes direction into a down-trend and adjust the strategy based on this. The option data is a powerful tool to exploit information about future expectation of market participants and adapt the strategy before it is too late. This is a big advantage compared to the vanilla models as they first have to experience bad performance by the assets before they will consider changing their strategy.

For comparing the performance of the (contextual) UCB and (contextual) TS on picking portfolios I consider 17 randomly generated portfolios. I choose 17 portfolios to stay close to the individual stock picking experiment. In this case both for the individual stock picking and portfolio picking, we have 17 arms in total. Later I will also investigate what happens for different amount of portfolios. A potential benefit of using portfolios is that you require less arms to be able to invest in all assets. However, a downside might be that if you do not generate enough portfolios the randomness in the portfolio weights might only generate bad performing portfolios.

Figure 3 shows the performance of the UCB and TS algorithms for picking portfolios. The performance of the algorithms is shown for the different values of the hyperparameters.

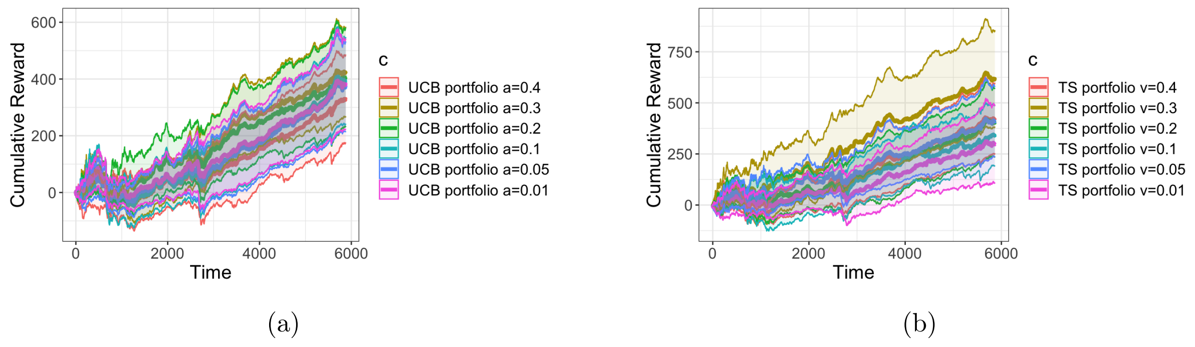
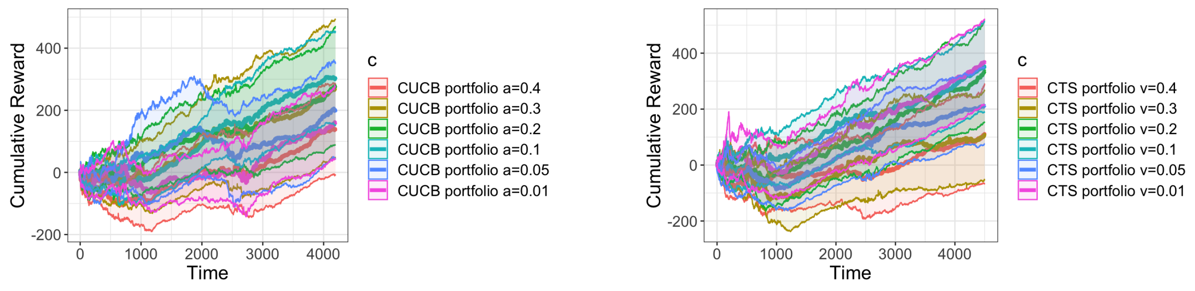


Figure 3: This plot shows the performance of the UCB algorithm (panel a) and the TS algorithm (panel b) for different values of the hyperparameters for portfolio picking. Simulation size $N = 100000$ *and number of simulation* $M = 14$.

The UCB algorithm performs best for $\alpha = 0.3$. Only $\alpha = 0.4$ performs substantially worse, while the other parameter values perform very similarly. The UCB and CUCB algorithms for individual stock picking also performed best for this value of the hyperparameter (when accounting for the difference in time horizon). The TS algorithm performs best for $\nu = 0.3$. In this case there is a substantial difference in performance by the algorithm compared to other values of ν. For all the other values the algorithm performs considerably worse, making this quite sensitive to the choice of the hyperparameter. For the individual stock picking experiment the optimal choice for the hyperparameter was 0.05, making the optimal value considerably different. Although in the case of the individual stock picking experiment the algorithm also performed relatively well for $\nu = 0.3$. However, this does indicate that the TS algorithm is more sensitive to changes in the hyperparameter compared to the UCB algorithm.

Figure 4 shows the performance of the CUCB and CTS algorithms for picking portfolios. As before, the performance is shown for different hyperparameter settings.



(a) (b)

Figure 4: This plot shows the performance of the CUCB algorithm (panel a) and the CTS algorithm (panel b) for different values of the hyperparameters for portfolio picking. Simulation size $N = 100000$ *and number of simulation* $M = 14$.

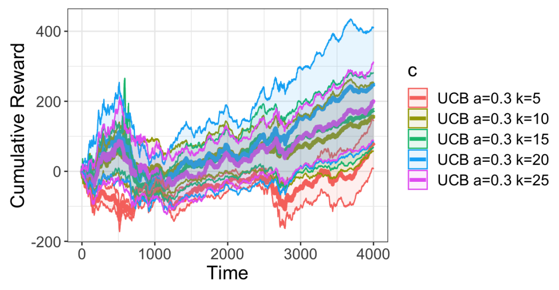
The CUCB algorithm performs best for $\alpha = 0.1$. The performance of $\alpha = 0.2, \alpha = 0.3$ are slightly worse. The other values for α perform substantially worse. This is a different optimal value compared to the (contextual) UCB for individual stock picking and the UCB for portfolio picking. However, the performance of the optimal value for those experiments ($\alpha = 0.3$) is also good in this case. This again indicates that the UCB algorithm is fairly robust against these different settings.

The CTS algorithm performs best for $\nu = 0.001$. However, the performance of $\nu = 0.1, \nu = 0.2$ is only slightly worse. The algorithm does perform decent for $\nu = 0.05$, but considerably worse for the other values. This is again a different optimal value compared to the other experiments. Compared to the UCB algorithm the choice of the hyperparameter for the TS algorithm is much more sensitive to the different settings.

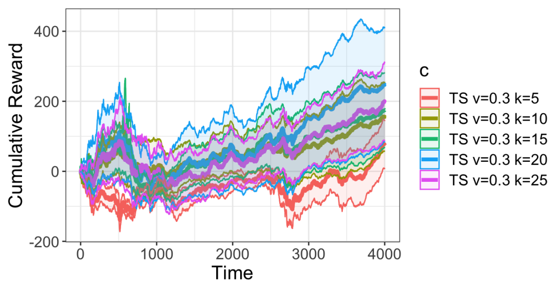
4.2 How many portfolios should we use?

In this section I investigate how sensitive the portfolio picking algorithms are to the choice of the amount of randomly generated portfolios. To this end, I simulate 25 portfolios randomly as described in Section 2.4. From these 25 portfolios I randomly select k portfolios. For the parameter settings I use the optimal hyperparameters that were found in the previous section. Note, that the optimal hyperparameters were found using 17 portfolios, meaning that the setup with close to 17 portfolios does have an advantage here.

Figure 5 shows the performance of the UCB (panel a) and TS (panel b) algorithms for different values of k (amount of available portfolios). For the UCB I set $\alpha = 0.3$ and for the TS I set $\nu = 0.3$, which were the optimal found hyperparameter in the previous section.
For both the UCB and TS algorithms the optimal amount of portfolios is 20. Notably, for both the UCB and TS the other values of k perform quite similar and not much worse



(a)



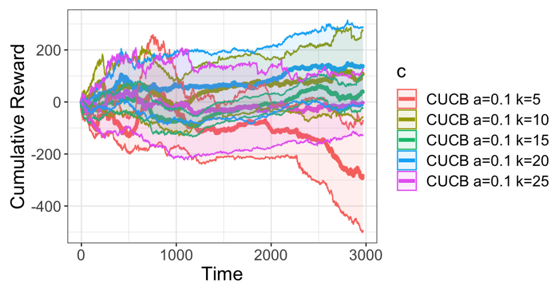
(b)

Figure 5: This plot shows the performance of the UCB algorithm (panel a) and the TS algorithm (panel b) for different amounts of portfolios. Simulation size $N = 100000$ and number of simulation $M = 14$.

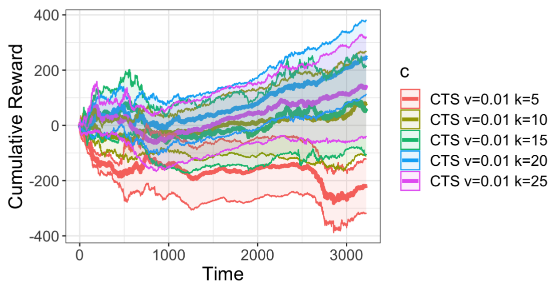
except for the case of $k = 5$. For $k = 5$ the performance drops drastically and the return is even negative for a large part of the sample path. A possible explanation is that 5 portfolios is too little and due to randomness in the portfolio weights do not contain enough good performing portfolios.

I have also tested this for other hyperparameter settings, but omit this for the sake of space. For other hyperparameter settings similar results were obtained. For the UCB $k = 15$ outperformed $k = 20$ once for $\alpha = 0.1$. Also for the TS $k = 15$ outperformed $k = 20$ once for $\nu = 0.1$. However, in both cases the difference was small and $k = 20$ performed well for all hyperparameter settings.

Figure 6 shows the performance of the CUCB (panel a) and CTS (panel b) algorithms for different values of k. The hyperparameter settings are $\alpha = 0.1, \nu = 0.01$.



(a)



(b)

Figure 6: This plot shows the performance of the UCB algorithm (panel a) and the TS algorithm (panel b) for different amounts of portfolios. Simulation size $N = 100000$ and number of simulation $M = 14$.

For the CUCB and CTS the optimal amount of portfolios is 20. This is the same results as for the vanilla UCB and TS. Again, we can see that the performance for $k = 5$ is bad, in this case even ending up substantially negative.

As for the vanilla UCB and TS, I also tested this for other values of the hyperparameters.

Again, I found that in all cases $k = 20$ performed best or at least close to the best performing value of k.

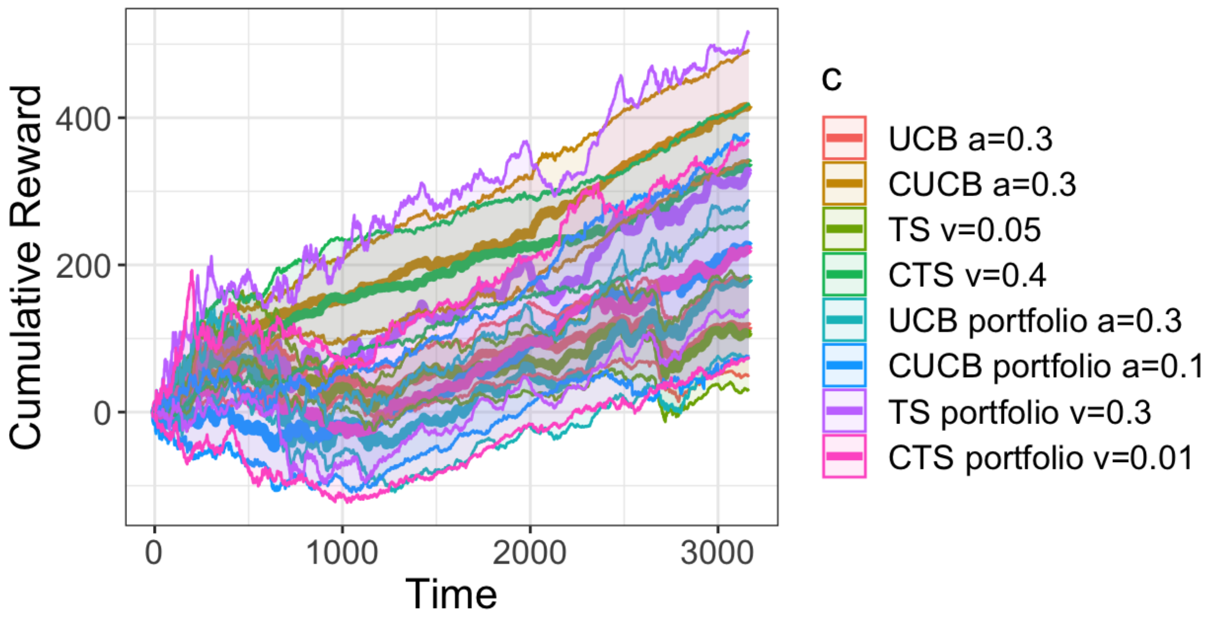
I conclude that the choice of k is fairly robust against the hyperparameter settings. It seems to be important to have at least enough portfolios to choose from, since for $k = 5$ the performance is always bad. However, too much is also not good as the performance for $k = 25$ is always worse than $k = 20$. This can possibly be explained by the fact that too many arms makes it more complicated for the model to find the optimal arms.

4.3 Comparison of the best methods

In Section 4.1 I investigated what the best hyperparameter settings are for the different algorithms and how sensitive they are to different choices. In Section 4.2 I investigated how many randomly generated portfolios we should use. In this section I combine the insights of the previous 2 sections to investigate which methods yields the highest returns. I use the optimal found hyperparameters and 20 randomly generated portfolios.

Figure 7 shows the performance of the different methods. The figure shows that the

Figure 7: Figure showing the performance of the different algorithm for the hyperparameters found to be optimal.



best performing algorithm is the CUCB for picking individual stocks. In second and close third we have the CTS for individual stocks and the TS for portfolios, respectively. Next up we have a very similar performance by the CTS for portfolios and CUCB for portfolios. Slightly below we have the UCB for portfolios. And at the bottom we have the TS and UCB for individual stocks.

For the TS algorithm for portfolios the performance drop substantially when including contextual variables. For the UCB algorithm for portfolios the performance increases slightly. This is surprising as for the individual stock algorithms the performance increases drastically when including the contextual variables into the algorithm. A possible explanation is that these contextual variables do not translate well into the portfolios context. The implied volatility is computed as a weighted average of the implied volatility of the individual stocks. It could be that this takes out too much of the signal that is in the implied volatility. However, it could also be that the relation is more non-linear for the portfolio case compared to the individual stock case.

We do see a large improvement for the vanilla UCB and TS for portfolios compared to the vanilla UCB and TS for individual stock picking. Traditional finance literature tells us that there is much more volatility in individual stocks compared to portfolios. Likely, due to too much noise the algorithms are not able to pick good individual stocks. Also, due to the non-stationary nature of stocks, over time the best stocks change. When investing in portfolios this risk of being invested in the wrong stocks becomes less as you are diversified over all stocks. Therefore, the noise and non-stationarity is less of a problem for picking portfolios.

The best performing method is the CUCB for individual stock picking. For a large part of the sample it performs very similarly to the CTS for individual stocks. However, starting at around $t = 2000$ the CUCB starts to outperform the CTS. Remarkably, these two algorithms are the only ones who do not suffer from large draw-downs. Likely, they are able to use the contextual variables to change course ahead of time as was already discussed in Section 4.1. For the portfolio algorithms this does not hold. Both the contextual and vanilla portfolio algorithms suffer from large draw-downs. This is additional evidence that the contextual variables do not generalize well to the portfolio level or that the linear relationship is not well suited.

5 Limitation and Extensions

I am working with randomly generated portfolios. To get more insight into the general performance of such a method it could be beneficial to look at a Monte Carlo study. This can be performed by simulating N runs, where in each run you simulate random portfolios and get the performance for these portfolios. Then after N runs you can take an average over the runs. This could give insight into the general performance of such methods, as they are run dependent due to the randomness. Additionally, a Monte Carlo study could give insight into the variance of the methods, which is especially important for portfolio management.

The inclusion of contextual variables such as implied volatility, OBV, and RSI has shown to improve the performance of the linear bandit algorithm for individual stocks. For further research other contextual variables can be included. For example, more stock specific information can be included such as company size, news sentiment, or earnings. This method can be combined with other advanced machine learning methods such as natural language processing for processing company specific news. The inclusion of contextual variables did not have the same effect at the portfolio level. For the UCB algorithm it improved the performance slightly, while for the TS algorithm it even decreased the performance. It could be interesting to look at other contextual variables for the portfolio algorithms.

The linear bandit algorithm used in this study can be extended to more complex models, such as neural networks or deep learning algorithms, which may better capture the non-linear relationships between contextual variables and optimal stock/portfolio choices. For the portfolio level it could be that the relationship between the contextual variables and the returns is non-linear and could not be captured by the linear bandit. This would explain why the contextual algorithms did not perform well at the portfolio level. However, such models may require larger datasets and more computational resources, which could limit their applicability in practice.

One of the problems in stock markets is that all models suffer from historical bias due to the non-stationary nature of stock prices. Therefore, there is no guarantee that these methods will work well in the future. Another factor that should be considered when such a strategy is implemented in the real stock market is the effect of transaction costs. For further research it is insightful to see how much transactions effect the results of the methods discussed in this paper.

Simulating portfolio weights from a normal distribution assumes that the portfolio weights follow a normal distribution, which may not always be the case. In reality, the distribution of portfolio weights may be skewed or have heavy tails, and using a normal distribution may not accurately capture this behavior. This could potentially lead to biased results and suboptimal portfolio weights. Instead, one approach could be to use a non-parametric method, such as bootstrapping or Monte Carlo simulations, to generate a distribution of possible portfolio weights. This would allow for more flexibility in capturing the true distribution of portfolio weights and could lead to more accurate results. Another approach could be to use machine learning methods to model the distribution of portfolio weights based on historical data and contextual variables.

6 Conclusion

In this paper, we examined whether incorporating option data, historical price action, and volume can improve the performance of the UCB and TS algorithms at both the individual stock and portfolio level. In addition, we introduced a novel approach to portfolio construction by randomly generating portfolios and allowing the algorithms to determine which are optimal.

Our findings suggest that the use of implied volatility, on-balance volume, relative strength index, and previous day returns as contextual variables can be valuable at the individual stock level. These variables helped the model avoid significant drawdowns, resulting in substantial outperformance compared to other methods. This is particularly useful as it allows the algorithms to perform well at all time points, without requiring long periods to generate profits. However, at the portfolio level, we did not observe a similar effect. This suggests that the contextual variables do not generalize well to the portfolio level, or their relationship may be non-linear, rendering the linear bandit models unsuitable. Nevertheless, we did find that vanilla algorithms perform considerably better at the portfolio level than the individual stock level, providing evidence for their usefulness in portfolio management. Future research may explore alternative contextual variables at the portfolio level and non-linear models for contextual variables.

We also conducted extensive sensitivity analysis to assess the impact of hyperparameters on the model's performance and the number of randomly generated portfolios. We found that the TS algorithm is more sensitive to hyperparameter selection than the UCB algorithm in this setting. Furthermore, we determined that 20 randomly generated portfolios were optimal, although values near this performed well. The only scenario that performed poorly was using only five portfolios, emphasizing the importance of utilizing a sufficient number of portfolios.

Overall, our results provide insights into the effectiveness of using contextual variables and random portfolio generation in portfolio management, highlighting the importance of carefully selecting hyperparameters and generating an adequate number of portfolios.

References

Liu, X.-Y., Xiong, Z., Zhong, S., Yang, H., and Walid, A., 2018. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*,

Sun, S., Wang, R., and An, B., 2021. Reinforcement learning for quantitative trading. *arXiv preprint arXiv:2109.13851*,

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, 142–158.

Yang, H., Liu, X.-Y., Zhong, S., and Walid, A. Deep reinforcement learning for automated stock trading: an ensemble strategy. In: *Proceedings of the first acm international conference on ai in finance*. 2020, 1–8.