Indian Institute of Technology Bombay

CS 337 Project Report

Deep Ensemble Reinforcement Learning for Adaptive Algorithmic Trading Strategies

Om Godage
21d100006

Shubham Hazra
210100143

Vijay Balsubramaniam
21d180043

supervised by

Prof. Preethi Jyothi

Contents

1 Introduction

In the dynamic realm of finance, adapting trading strategies to unpredictable markets is paramount. Our project delves into Deep Ensemble Reinforcement Learning [1], comparing its performance to that of existing statistical methods, and exploring its potential for adaptive algorithmic trading strategies.

We model the stock data as a Markov Decision Process, as implement an online learning algorithm to outperform the market. We also explore the use of deep learning to model the stock data, and compare the performance of ensemble with that of a single deep learning model.

2 Statistical Models

The statistical models we have implemented are as follows:

- **Hierarchical Risk Parity** - A portfolio optimization technique that clusters assets into groups, and assigns weights to assets based on their risk contribution to the portfolio.
- **CVaR** - A risk measure that is a more robust alternative to VaR, and is used to minimize the risk of the portfolio. We use the CVaR as a constraint in the optimization problem.

Both the above models are implemented in the `MPT.ipynb` file, utilising the library **PyPortfolioOpt**.[2]

2.1 Hierarchical Risk Parity

Hierarchical Risk Parity optimally allocates investment risk across different asset classes by considering both the hierarchy of portfolio constituents and their respective risk contributions. The algorithm is as follows:

Algorithm 1: Hierarchical Risk Parity Algorithm

```
Data: covariance_matrix, expected_returns
Result: asset_weights
1 distance_matrix ← 0.5 (1−$correlation\_matrix$)
2 clusters ← hierarchical_clustering(distance_matrix);
3 inv_distance_matrix ← inv(distance_matrix);
4 hierarchy_weights ← compute_hierarchy_weights(inv_distance_matrix, clusters);
5 asset_weights ← compute_asset_weights(hierarchy_weights, expected_returns);
```

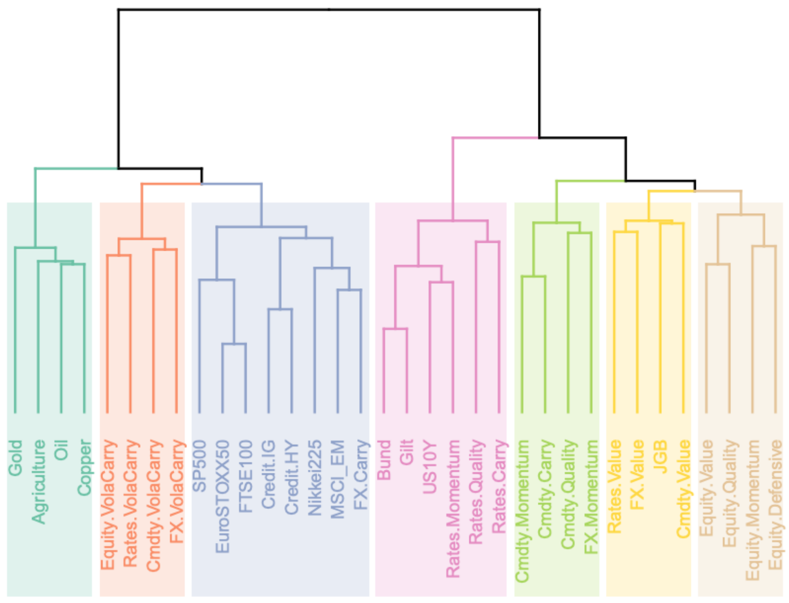

Figure 1: Hierarchical Risk Parity

The algorithm forms a binary tree of assets, where the leaves are the assets themselves. The root node is the entire portfolio. The distance between two assets is defined as 1−correlation_matrix. The distance between two clusters is defined as the average distance between all the assets in the two clusters. The distance between a cluster and an asset is defined as the average distance between the asset and all the assets in the cluster.

2.2 Conditional Value at Risk

Conditional Value at Risk (CVaR) is a risk measure that is a more robust alternative to Value at Risk (VaR). It is defined as the expected loss given that the loss is greater than the VaR.

$$CVaR(\omega, \beta) = \frac{1}{1-\beta} \int_{L(w,r) \geq \alpha(w)} L(w,r) \cdot p(r) dr$$

where

- w for the vector of portfolio weights
- r for a vector of asset returns (daily), with probability distribution $p(r)$
- $L(w,r) = -w^T r$ for the loss of the portfolio
- α or the portfolio value-at-risk (VaR) with confidence β

We use CVaR as a constraint in the optimization problem, to minimize the risk of the portfolio.

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & CVaR(w, \beta) \\ \text{subject to} \quad & \mu^T w \geq \mu_{min} \\ & w^T \Sigma w \leq \sigma_{max}^2 \\ & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\} \end{aligned}$$

where w is the weight vector, μ is the expected return vector, Σ is the covariance matrix, μ_{min} is the minimum expected return, σ_{max} is the maximum risk, and n is the number of assets.

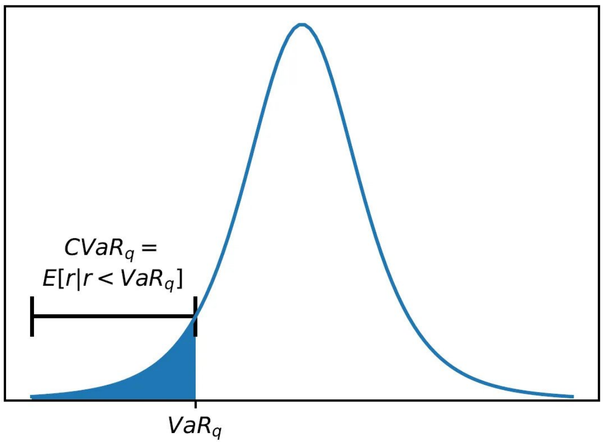

Figure 2: Pictorial representation of CVaR

3 Reinforcement Learning Models

The reinforcement learning models have been imported from the `stable_baselines3` library. We used the models:

- **A2C** - Advantage Actor Critic
- **PPO** - Proximal Policy Optimization
- **DDPG** - Deep Deterministic Policy Gradients

3.1 Advantage Actor Critic

Advantage Actor Critic (A2C)[3] is an on-policy algorithm that uses the advantage function to update the policy. The advantage function is defined as the difference between the Q-value and the value function.

$$A(s, a) = Q(s, a) - V(s)$$

Where Q is the bootstrapped value function, and V is the value function.

$$Q(s, a) = \mathbb{E}[r + \gamma V(s')]$$

Algorithm 2: Advantage Actor Critic

```
   Data: Environment, Actor and Critic neural networks, Learning rates $\alpha_w, \alpha_\theta$,
         Discount factor $\gamma$
 1 Initialize $s, \theta, w$ with random weights;
 2 Sample a $\sim \pi_\theta$(s‖a)
 3 for $t$ = 1 ... $T$ do
 4     Sample reward $r_t \sim$ R(s, a) and next state $s' \sim$P(s'‖s, a);
 5     Sample next action $a' \sim \pi_\theta(s'\|a')$;
 6     Update the policy parameters $\theta$ using the advantage function;
 7     $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \log \pi_\theta(a\|s) A(s,a)$;
 8     Compute the TD error for time step t;
 9     $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$;
10     and use it to update the action value parameters;
11     $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$;
12     Move a to a';
13     Move s to s';
14 end
```

3.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an on-policy algorithm, derived from the TRPO algorithm, that uses the clipped surrogate objective function to update the policy. The clipped surrogate objective function is defined as:

$$L^{CLIP}(\theta) = \mathop{\mathbb{E}}_{t}\left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Where $r_t(\theta) = \frac{\pi_\theta(a_t\|s_t)}{\pi_{\theta_{old}}(a_t\|s_t)}$ is the probability ratio, $\hat{A}_t$ is the advantage function, and ϵ is the clipping parameter.

$$\hat{A}_t = \frac{A_t - \mu(A_t)}{\sigma(A_t)}$$

Where $\mu(A_t)$ and $\sigma(A_t)$ are the mean and standard deviation of the advantage function. The clipping parameter is used to prevent the policy from changing too much, thus improving stability.
This objective implements a way to do a Trust Region update which is compatible with Stochastic Gradient Descent, and simplifies the algorithm by **removing the KL penalty** and need to make adaptive updates.

3.3 DDPG

Deep Deterministic Policy Gradient (DDPG)[4] is an off-policy algorithm that uses the actor-critic method to learn the policy and the value function. The actor network is used to learn the policy, and the critic network is used to learn the value function. The actor network is updated using the gradient of the Q-value with respect to the policy parameters, and the critic network is updated using the TD error.

$$\nabla_\theta J \approx \mathbb{E}[\nabla_\theta \mu_\theta(s) \nabla_a Q(s,a)|_{s=s_t, a=\mu_\theta(s_t)}]$$

$$\delta_t = r_t + \gamma Q(s', \mu_\theta(s')) - Q(s,a)$$

DDPG uses **Experience Replay** and **Target Networks** to improve stability.

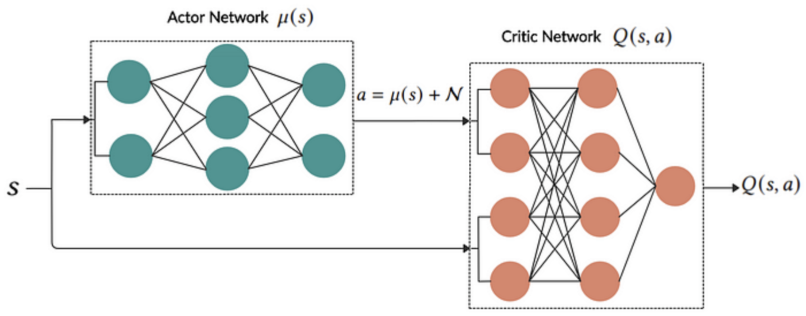

Figure 3: DDPG

4 Code

4.1 Ensembling

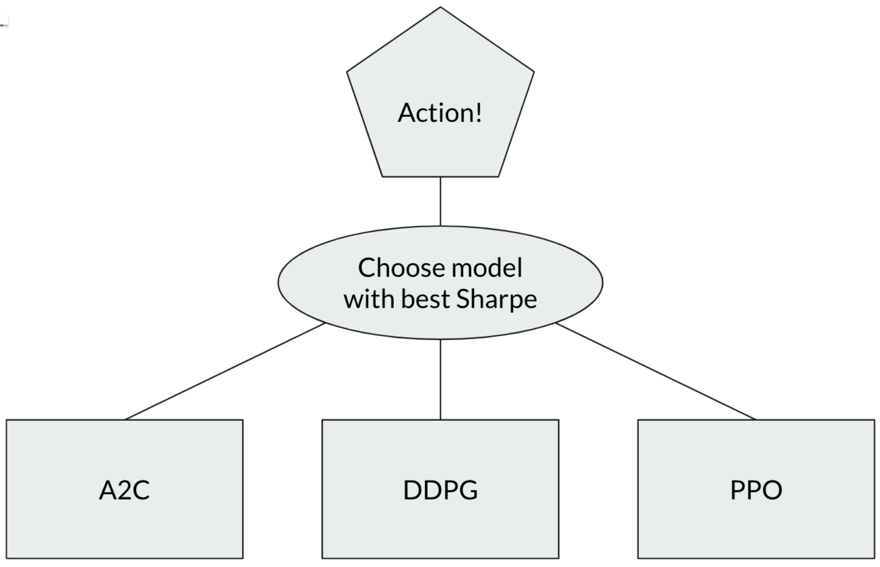

Figure 4: Ensembling method

The ensemble model is implemented in the `model.py` file. The ensemble model breaks the data into multiple windows, and trains all models on each window. The sharpe ratio

of each model is calculated, and the model with the highest sharpe ratio is used to predict the next action. The ensemble model is trained on a part of the dataset, and then is used for validation, learning online.
The Sharp Ratio is defined as:

$$\text{S} = \frac{\mathbb{E}[R_a - R_b]}{\sigma_a}$$

where R_a is the return of the asset, R_b is the risk free return, and σ_a is the standard deviation of the asset.

4.2 Training the Agent

4.2.1 Data

We track and select the **Dow Jones 30** stocks and use historical daily data from 01/01/2010 to 01/10/2021 to train the agent and test the performance. We used the library **yfinance** to obtain this data from Yahoo Finance.

4.2.2 MDP model for stock trading

The environment is an MDP using the OpenAI Gym API.
The **action space** is discrete, with $2 \times h_{max} + 1$ actions, where h_{max} is the maximum number of shares that can be bought or sold, and ranges from $[-h_{max} \dots 0 \dots h_{max}]$, although we normalize this data, to lie in the range $[-1, 1]$
The **state space** s is a vector, including the stock prices, stock shares and balance. We store some technical indicators, like MACD, RSI, etc. which are computed by the library **stockstats**.
The **reward function** is defined as:

$$r_t = \Delta\text{portfolio\_value} + \Delta\text{Balance}$$

The transitions for each state are one of three types:

- **Buy** - Buy shares of an asset
- **Sell** - Sell shares of an asset
- **Hold** - Do nothing

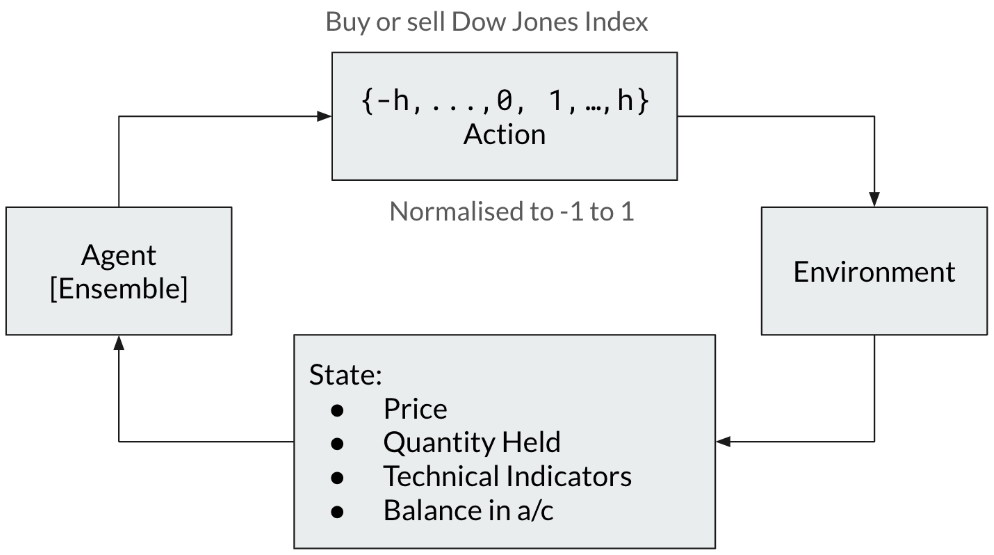

Figure 5: Model training

5 Results

5.1 Index comparison

In the figures below, we show how our ensemble strategy behaves on different market indices, specifically on Dow Jones, DAX and TecDAX.

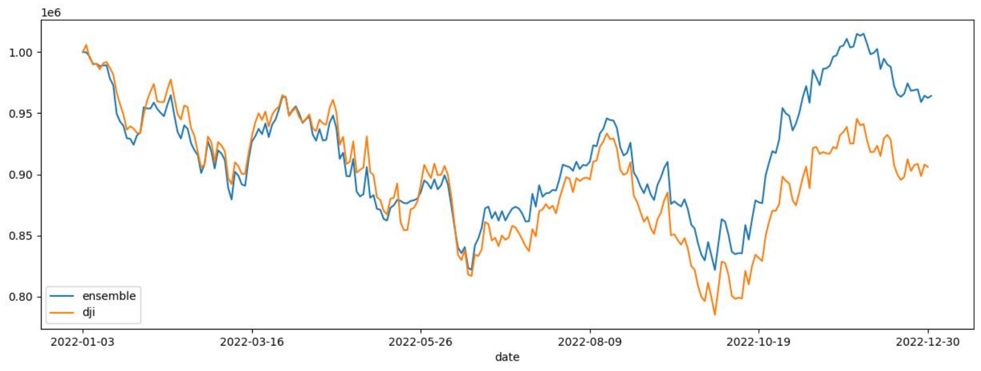

Figure 6: Ensemble with Dow Jones

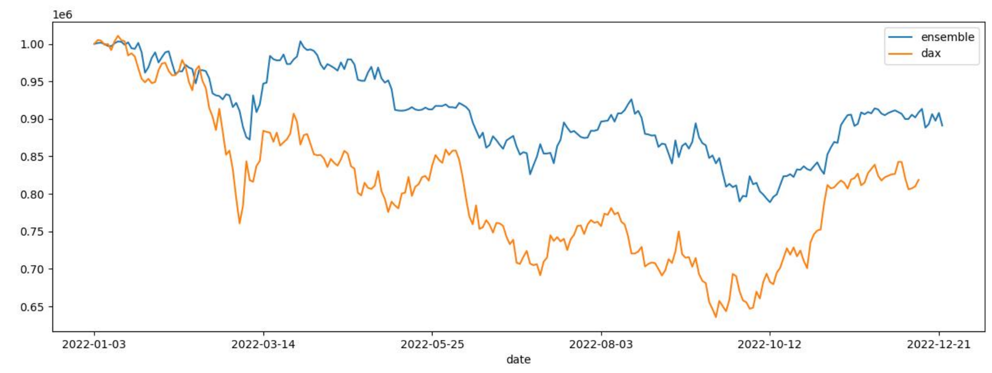

Figure 7: Ensemble with DAX

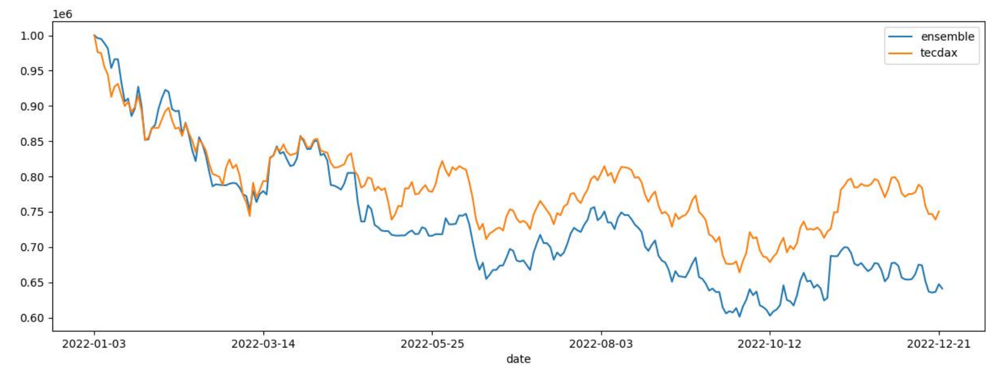

Figure 8: Ensemble with TecDAX

We can see that our model (blue) outperforms the DAX and the Dow Jones index, but fails to break even with the TecDAX index, which could be due to the model being trained on companies of various sectors, but TecDAX only considers German companies from the technology center, which also displays the **poor generalizability** of our model.

5.2 Comparing other methods

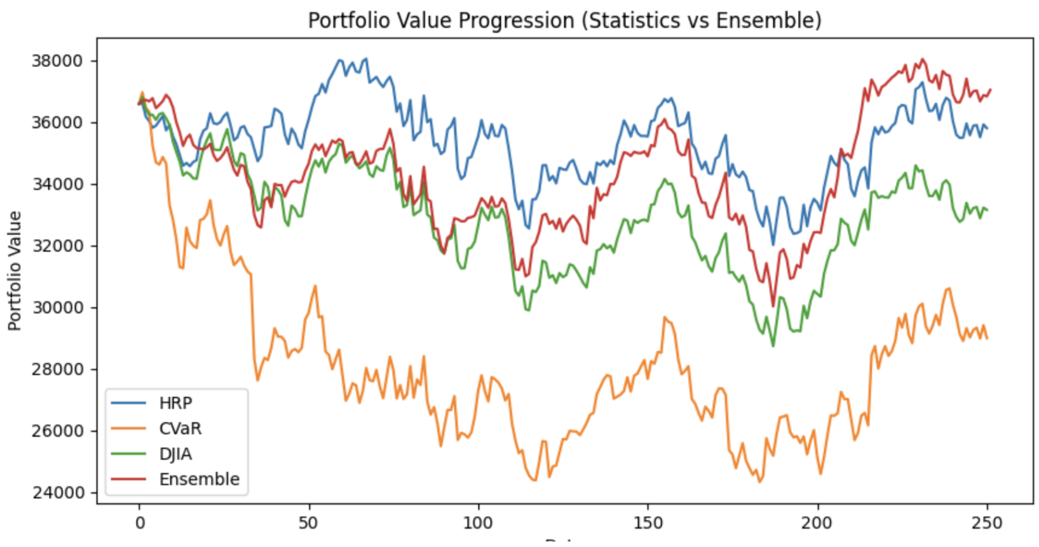

Figure 9: Ensemble vs. Statistical

There is quite a competition between the HRP and our Ensemble method, although our method emerges on the top at the end. Also, the ensemble method clearly overpowers CVaR, which even underperforms the market index.

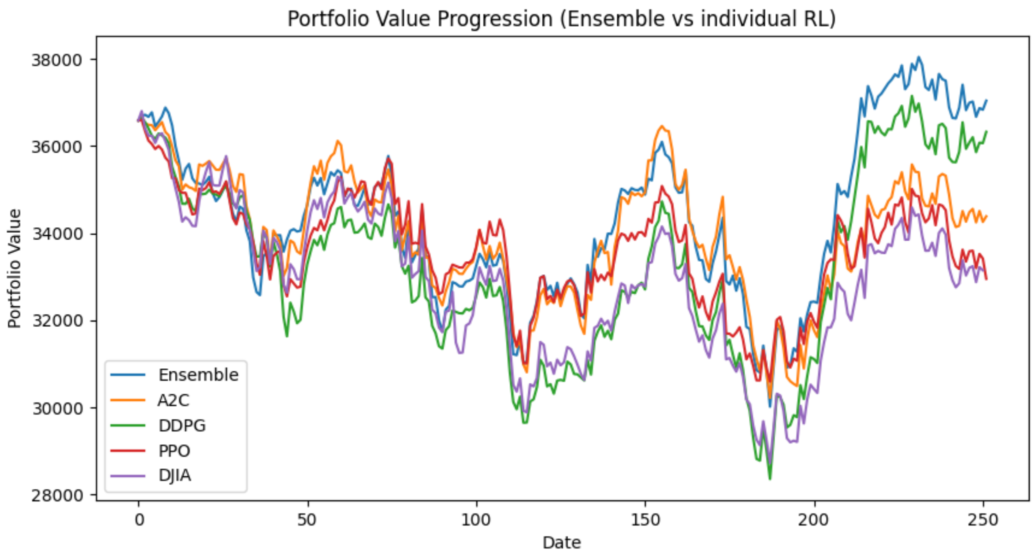

Figure 10: Ensemble vs. single RL algos

Almost throughout the test data, the ensemble method is seen to be performing better than the individual (A2C, DDPG, PPO) algorithms, which also proves that our sharp ratio maximising strategy seems to work well

6 Hinderances and Loopholes

Some things we noticed that we could work on further were:

1. **Insufficient Technical Indicators**: While technical indicators provide valuable information about historical price movements, they may not capture the full complexity of stock price dynamics. Many useful data is lost, which leads us to the second point
2. **Markov Decision Process?**: The Markov Property assumes that the future state depends only on the current state, not the sequence of states leading up to it, due to technical indicators, although as in point 1, a lot of the data is lost, which makes us question this assumption. In the stock market, the Markov Property is often violated due to long-term dependencies and memory effects. Prices can be influenced by past events and trends, requiring models that can capture longer-term dependencies.
3. **Won't most good stocks in the Dow Jones rise anyway?**: We saw that our model was quite sensitive to the general trend of the training model, and we encountered a point where the model was simply buying stock every time until it ran out of balance, as the training data had an overall upward trend. Neural networks can indeed learn patterns and trends, but relying solely on past trends might not be sufficient for predicting future stock prices.
   One solution to mitigate this problem would be to perform **detrending** which removes the trend component from the time series data. This can help the model focus on the underlying patterns rather than capturing the overall trend.
4. **Features and Market sentiments**: Long-term investments require considering fundamental factors such as financial statements, earnings reports, and economic indicators. Incorporating these features into the model, along with technical indicators, can provide a more holistic view of a company's health and potential for long-term growth. Also, news and social media, plays a significant role in stock price movements. Sentiment analysis of financial news and social media data can provide valuable insights.

7 Conclusion

In summary, the Ensemble model exhibits competitive performance, but the observed variations over time suggest that RL algorithms may not be fully adapted to handle the complexity and dynamics of financial markets consistently.
To enhance adaptability and robustness, further refinement in feature selection and the exploration of diverse ensemble architectures is imperative. This evolution aims to capture complex market patterns and fortify the model's resilience across diverse market conditions.

References

[1] Liu, X.-Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., and Wang, C. D. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance, 2022.

[2] Martin, R. A. Pyportfolioopt: portfolio optimization in python. *Journal of Open Source Software 6*, 61 (2021), 3066.

[3] Simonini, T. Advantage actor critic (a2c), 2022. Accessed: November 28, 2023.

[4] Yoon, C. Deep deterministic policy gradients explained, 2019. Accessed: November 28, 2023.