ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

A COMPARISON ANALYSIS OF DIFFERENT ASSET ALLOCATION TECHNIQUES

FIN-525: FINANCIAL BIG DATA

Ana Lucia Carrizo Delgado

Nizar Ghandri

January, 2022

ABSTRACT

The goal of this project is to apply the concepts learned in the Financial Big Data course taught at EPFL to compare different asset allocation methods. In particular, when building a solid portfolio model, we need robust estimates, that are less sensitive to noise and outliers. The data we used consisted of historical prices of the companies that form the S&P500. Classical portfolio optimization methods such as equal weights, market value, minimum variance, and mean variance were implemented. Furthermore, we used reinforcement learning to perform asset allocation.

CHAPTER 1

FOUNDATIONS OF PORTFOLIO THEORY

Before we begin describing the different asset allocation techniques we implemented, we need to first go through some basic mathematical definitions in which our project will be based.

1.1 RETURNS

We define the return on a stock from time $t-1$ to time t as:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

The expected returns of the individual assets are: $E[r_i] = \mu$.

Thus, the total expected return of portfolio P can be written as:

$$\mu_P = E[r_P] = \sum_{i=1}^{n} w_i E[r_i] = \sum_{i=1}^{n} w_i \mu_i$$

where w_i is the weight that stock i holds in the portfolio.

1.2 LOG-RETURNS

For small price variations, log-returns are first order equivalents of returns. In our project, we decided to work with log-returns instead of simple returns given some nice properties of the logarithmic function. For example, log(1+return) is a Normal Distribution which makes it a better fit to assumptions of lots of stochastic pricing models, it is also symmetric as its range is $(-\infty, \infty)$, finally, log(1+return) can add up to get cumulative return, whereas it does not apply for simple returns.

1.3 RISK

We consider risk to be the variance of returns. For a given asset i the risk is:

$$\sigma_i^2 = Var(r_i) = E[(r_i - \mu_i)^2]$$

The covariance of stocks i and j is:

$$Cov(r_i, r_j) = E[r_i r_j] - E[r_i]E[r_j]$$

The correlation of assets i and j is:

$$Corr(r_i, r_j) = \frac{Cov(r_i, r_j)}{\sqrt{\sigma_i^2 \sigma_j^2}}$$

CHAPTER 2

DATA EXPLORATION

2.1 DATA RECOLLECTION

The data consists of the stock prices from the companies that form the S&P 500 (502 and companies). We gathered data from 1st of January of 2012 till the 1st of January of 2022, from the yfinance module. This is the maximal amount of data we can collect from there. This yields 3650 dates for 500 comapnies, which means we have around 1825000 data points ! We also rely on the Wharton Research Data Service (wrds) database to collect the shares outstanding for each of the companies out there.

2.2 PREPROCESSING

For the data preprocessing we rely mainly on pandas given that modern RAM are pretty large and can fit the data in memory. We filled empty values with a forward fill, to maintain the causal property of time-series data. Similarly, we removed the companies that had no returns (NaNs) for a number of dates bigger than a specific threshold. In our case, we chose it to be 20% of the days. For the wrds database, we mainly rely on SQL queries to obtain the shares outstanding, using multiple joins and some functional programming techniques.

CHAPTER 3

METHODS

In this chapter we explore different asset allocation techniques. We use as baselines classical models such as the equally weighted portfolio, and the value weights portfolio. Furthermore, we compute another baseline model, the Mean Variance Portfolio (MVP), and implement it using an unfiltered covariance matrix and a bahc-filtered one. Additionally, we used reinforcement learning to create a neural network that allocates assets.

3.0.1 EQUALLY WEIGHTED PORTFOLIO

An equally weighted portfolio is such that every asset in the portfolio holds the same importance; in the sense that all stocks have the same weight.

3.0.2 MARKET VALUE-WEIGHTED PORTFOLIO

Contrarily to the Equally Weighted Portfolio, in a market value-weighted portfolio, the stocks with a higher market cap will receive a higher weighting in the portfolio. In essence, this means that smaller market cap companies will have less of an impact on the performance of the portfolio.

3.0.3 MEAN VARIANCE PORTFOLIO

Mean-Variance analysis is a one part of modern portfolio theory, in which an investor can compute the trade off between risk and reward expectation, resulting in an optimal portfolio.

Markowitz theory is based on two assumptions:

- Every investor's goal is to maximize returns for any level of risk.
- Risk can be reduced by diversifying a portfolio through individual, unrelated securities.

To find the optimal weights, we must look at the assets' historic returns and pairwise covariances. As stated above the objective is to minimize the variance for a given level of expected return. Mathematically, the Mean-Variance optimization problem can be formulated in the following way:

$$\begin{cases} \min_w w^T \Sigma w \\ w^T \mu = r_p \\ w^T \overrightarrow{1} = 1 \end{cases}$$

where:

- r_p is the target return
- w is the vector of stocks' weights
- Σ is the stocks' covariance matrix
- μ is the vector of asset returns

It is worth to notice the fact that our choice of portfolio does allow for shorting. If we only wanted a longing portfolio, we would have had to add an additional constraint on w which would transform the computational problem into a non-linear task.

In order to visualize better the risk/return trade-off, investors aid themselves using the efficient frontier. The portfolios in the efficient frontier have the lowest possible variance for their expected return, or vice-versa, the highest expected return for their variance.

The optimal weights for a given target return, at the efficient frontier, are computed with the following expression (Jiao 2003):

$$w\left(r_p\right) = \frac{\left(a\Sigma^{-1}\mu - b\Sigma^{-1}\overrightarrow{1}\right) r_p + \left(c\Sigma^{-1}\overrightarrow{1} - b\Sigma^{-1}\mu\right)}{ac - b^2}$$

$$a = \overrightarrow{1}^T \Sigma^{-1} \overrightarrow{1}$$

$$b = \mu^T \Sigma \overrightarrow{1}$$

$$c = \mu^T \Sigma \mu$$

THE TANGENCY PORTFOLIO

Although, the frontier is a good visualization tool, our objective of finding the optimal portfolio remains unsolved. This is where the tangency portfolio comes in. The tangency portfolio is meant to maximize expected risk-adjusted returns and is located at the point, where the Capital Allocation Line is tangent to our frontier curve. In other words, to find the tangency portfolio we maximize the Sharpe Ratio of the portfolio.

Mathematically, this means:

$$\max_w \frac{w^T \mu - r_f}{\left(w^T \Sigma w\right)^{-1/2}}$$

such that $w^T \overrightarrow{1} = 1$.

Which can be solved by using Lagrangian and derivatives (Zivot 2013), yielding:

$$w_{tan} = \frac{\Sigma^{-1}\left(\mu - r_f \overrightarrow{1}\right)}{\overrightarrow{1}^T \Sigma^{-1}\left(\mu - r_f \overrightarrow{1}\right)}$$

where w_{tan} is the vector that holds the weights of the tangency portfolio and r_f is the risk-free rate.

IMPLEMENTATION

Rolling windows were used in order to obtain the weights of the tangency portfolio. We decided to compare returns using windows of 90 and 180 days.

LIMITS OF THE MVP

The Mean-Variance portfolio is easy to understand and implement, it easily explains the relationship between risk and returns, and has proven to be a powerful tool in portfolio's performance analysis. Nevertheless, it has numerous limitations. Since the algorithm works in a way that allocates more weight to the "best" stocks, it can be greatly affected by slight inaccuracies and changes in returns. Thus, it is very unstable. We might loose the benefit of diversification if our portfolio has, for example, one asset dominating the portfolio.

As will be shown in the results, when using a classical covariance matrix, our results vary depending on the window size. This means that the covariance matrix is not robust. In the next section, we will describe how we solved this issue.

3.0.4 MEAN VARIANCE PORTFOLIO WITH BAHC FILTERING

> “*Nobody should be using the sample covariance matrix for the purpose of portfolio optimization.*” ”
>
> EPFL Course FIN-525, Lecture 10, Slide 3

MOTIVATION

When the number of objects is similar to the number of features, the covariance matrix becomes very noisy, which is the case with financial data. This is known as the curse of dimensionality. Numerous attempts to "clean" covariance matrices can be found in the literature. Most notoriously, the use of hierarchical clustering techniques has proven to be very effective when dealing with financial data.

As seen in the figure above (Fig. 3.1), our data is quite noisy. The classic covariance matrix is not able to properly capture structures.

BOOTSTRAPPED AVERAGE HIERARCHICAL CLUSTERING (BAHC)

The main objective of BAHC is to improve upon the classic hierarchical clustering method to capture more of the structure of the eigenvectors while increasing robustness. The idea is to compute filtered hierarchical structures of many bootstrapped copies of the initial data, which yields probabilistic hierarchical structures. In other words, BAHC does HCAL to bootstraps of the original data and then averages all HCAL-filtered matrices to obtain a new matrix (Bongiorno and Challet 2021).

We can easily improve upon the classical covariance matrix by applying HCAL (Fig. 3.2a), and even better, BAHC (Fig. 3.2b). Below, we can see the resulting matrices:

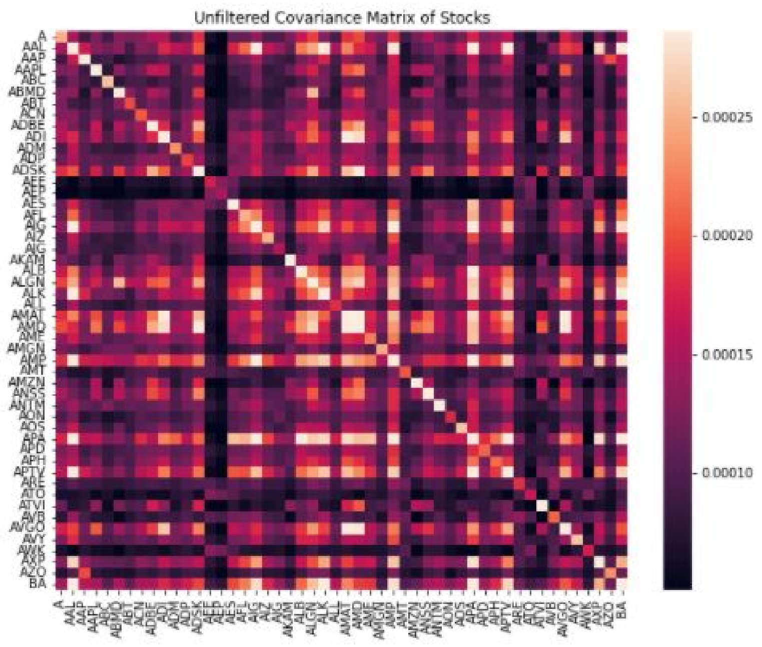
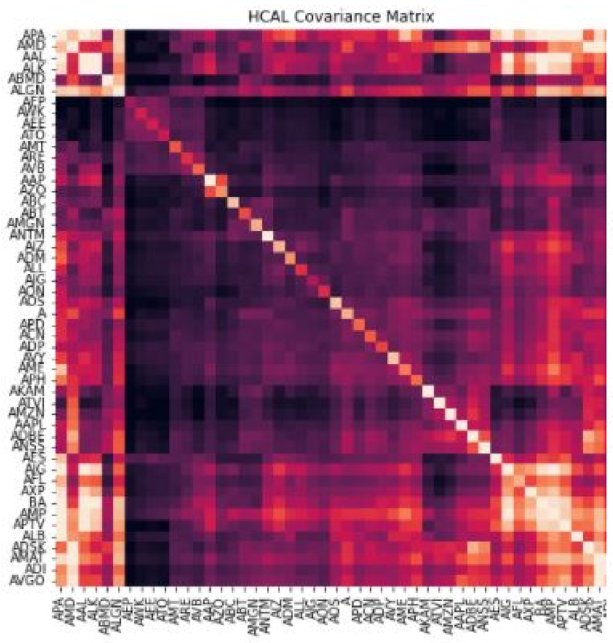


FIGURE 3.1
Covariance Matrix of first 50 companies in our data


(A) HCAL

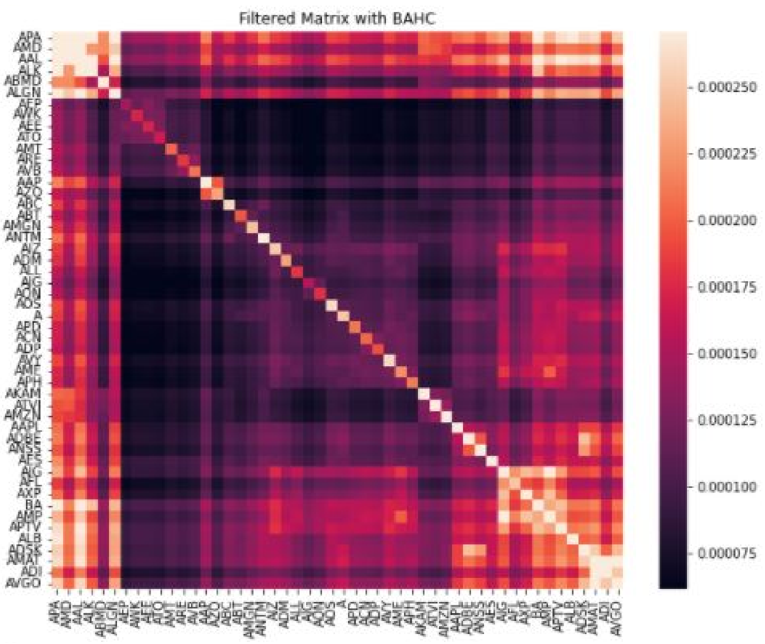

(B) BAHC

FIGURE 3.2
Filtered Covariance Matrices of first 50 companies in our data

IMPLEMENTATION

We implemented the Mean Variance Portfolio in the same way as with the simple covariance matrix. We chose to compute returns for a rolling window of 90 and 180, and compared results.

3.0.5 REINFORCEMENT LEARNING

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Reinforcement learning differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Partially supervised RL algorithms can combine the advantages of supervised and RL algorithms.

The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context use dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

Two functions are very important in Reinforcement learning the value function and the Q function.

According to the Sutton and Barto book, the V function is

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

and the **Q** function is

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

The reward R_t is always discounted by a factor γ for convergence issues (theoretically) and to give more importance to newer rewards rather than old ones.

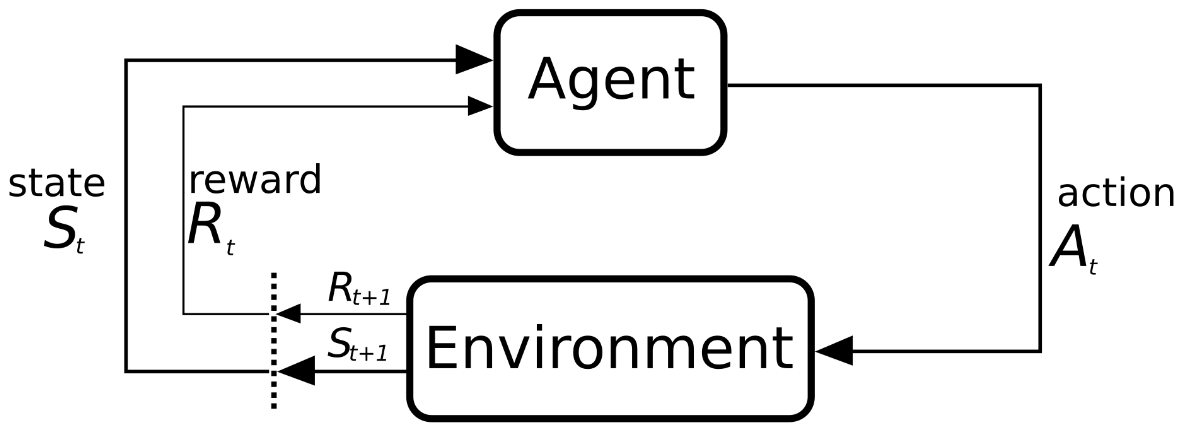


FIGURE 3.3
Reinforcement Learning

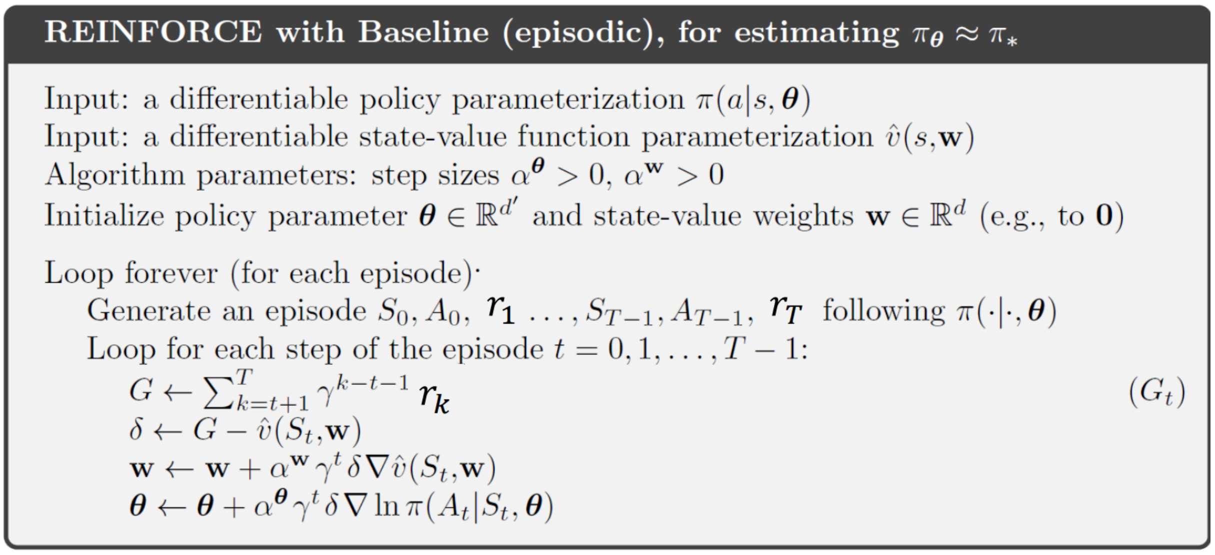
Here, in our case, the state S_t time corresponds to the log returns at time t, the action A_t corresponds to the the set of weights we are putting in each stock and finally the reward at time $t$is defined for simplicity to be:

$$R_t =< S_t, A_t >$$

where $< \cdot >$ is the scalar product between the two vectors.

ACTOR CRITIC: REINFORCE

Here is the pseudo algorithm of the REINFORCE model:



```
REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode)·
    Generate an episode $S_0, A_0, r_1 \ldots, S_{T-1}, A_{T-1}, r_T$ following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$                    $(G_t)$
        $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$
```

FIGURE 3.4
Reinforce with baseline, taken from Sutton and Barto book

Here we rely on deep learning (Multi layer perceptrons) to approximate to the differentiable policy and

and to generate the state-value estimator.

The reason we substract a baseline is that substracting the expectation provides estimates that have (normally) smaller variance (look less noisy). In our case the best estimation of the expected reward received in state S_t would be, by definition, $V(S_t)$ which is the value function of that state.

We use the cross entropy loss at the output layer defined by:

$$\mathrm{H}(p, q) = -\sum_x p(x) \log q(x)$$

as it provides weights that are strictly positive and sum up to one and allows the $ln(\pi(A_t|S_t, \theta))$ to appear in the update rule. It also provides an easy transition to the online update rule with the log likelihood trick as it allows us to keep a correct statistical weight when averaging a gradient in the online mode.

$$\begin{aligned}
\nabla_\theta J &= \int \nabla_\theta p(H) R(H) dH \\
&= \int \frac{p(H)}{p(H)} \nabla_\theta p(H) R(H) dH \\
&= \int p(H) \nabla_\theta \log p(H) R(H) dH
\end{aligned}$$

This model is called an on policy method. On-policy methods attempt to evaluate or improve the policy that is used to make decisions. In contrast, off-policy methods evaluate or improve a policy different from that used to generate the data. Deep Q learning, which we will explore in the following section, is an example of an off policy method.

DEEP Q-LEARNING

The agent here will perform the sequence of actions that will eventually generate the maximum total reward. This total reward is also called the Q-value and we will formalise our strategy as:

The above equation states that the Q-value yielded from being at state s and performing action a is the immediate reward $r(s, a)$ plus the highest Q-value possible from the next state s'.

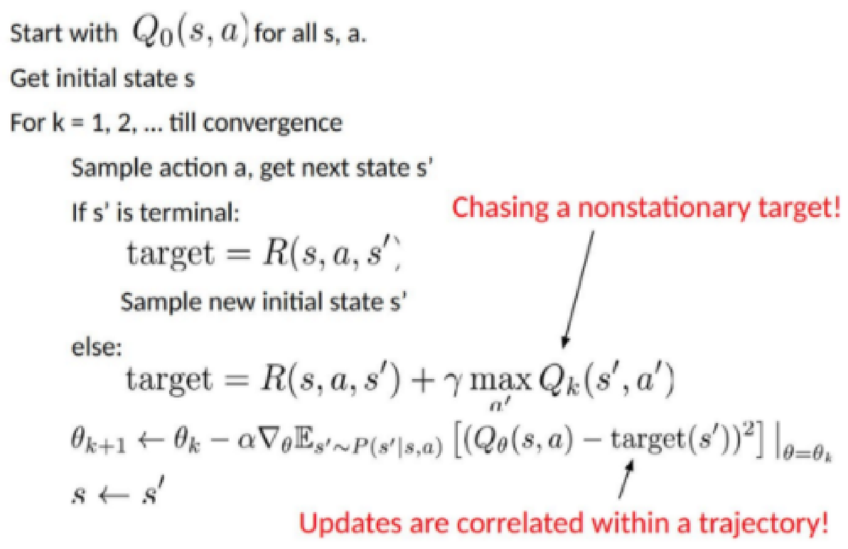
$$Q(s, a) = r(s, a) + \gamma \max_a Q\left(s', a\right)$$

Since this is a recursive equation, we can start with making arbitrary assumptions for all q-values. With experience, it will converge to the optimal policy. In practical situations, this is implemented as an update:

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

where α is the learning rate or step size. This simply determines to what extent newly acquired information overrides old information.

The full pseudo algorithm for DQN is:



```
Start with $Q_0(s,a)$ for all s, a.
Get initial state s
For k = 1, 2, ... till convergence
    Sample action a, get next state s'
    If s' is terminal:
        $\text{target} = R(s,a,s')$
        Sample new initial state s'
    else:
        $\text{target} = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$
    $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[ (Q_\theta(s,a) - \text{target}(s'))^2 \right] \big|_{\theta=\theta_k}$
    $s \leftarrow s'$
```

Chasing a nonstationary target!

Updates are correlated within a trajectory!

FIGURE 3.5
Deep Q learning

To prevent the non convergence of the gradient descent, because of the optimization on non stationary targets and the correlated updates within every gradient trajectory, we use two neural networks instead of one.

1. Select an action using the epsilon-greedy policy. With the probability epsilon, we select a random action a and with probability 1-epsilon, we select an action that has a maximum Q-value, such as a = $\text{argmax}(Q(s,a,w))$
2. Perform this action in a state s and move to a new state s'to receive a reward. This state s' is the preprocessed image of the next game screen. We store this transition in our replay buffer as $\langle s,a,r,s' \rangle$
3. Next, sample some random batches of transitions from the replay buffer and calculate the loss
4. Our loss function is:

$$L_\delta(a) = \begin{cases} \frac{1}{2}\left(r + \gamma \max_{a'} Q\left(s',a';\theta'\right) - Q(s,a;\theta)\right)^2 & \text{for } |a| \le \delta \\ \delta\left(\left|\left(r + \gamma \max_{a'} Q\left(s',a';\theta'\right) - Q(s,a;\theta)\right)\right| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases}$$

   which is known as the Huber loss,. It satisfies the properties of a statistically robust loss function, thus it's ideal for these kind off stochastic approximation methods
5. Perform gradient descent with respect to our actual network parameters in order to minimize this loss
6. After every C iterations, copy our actual network weights to the target network weights.
7. Repeat these steps for number of episodes

CHAPTER 4

RESULTS

4.1 EQUAL WEIGHTS PORTFOLIO

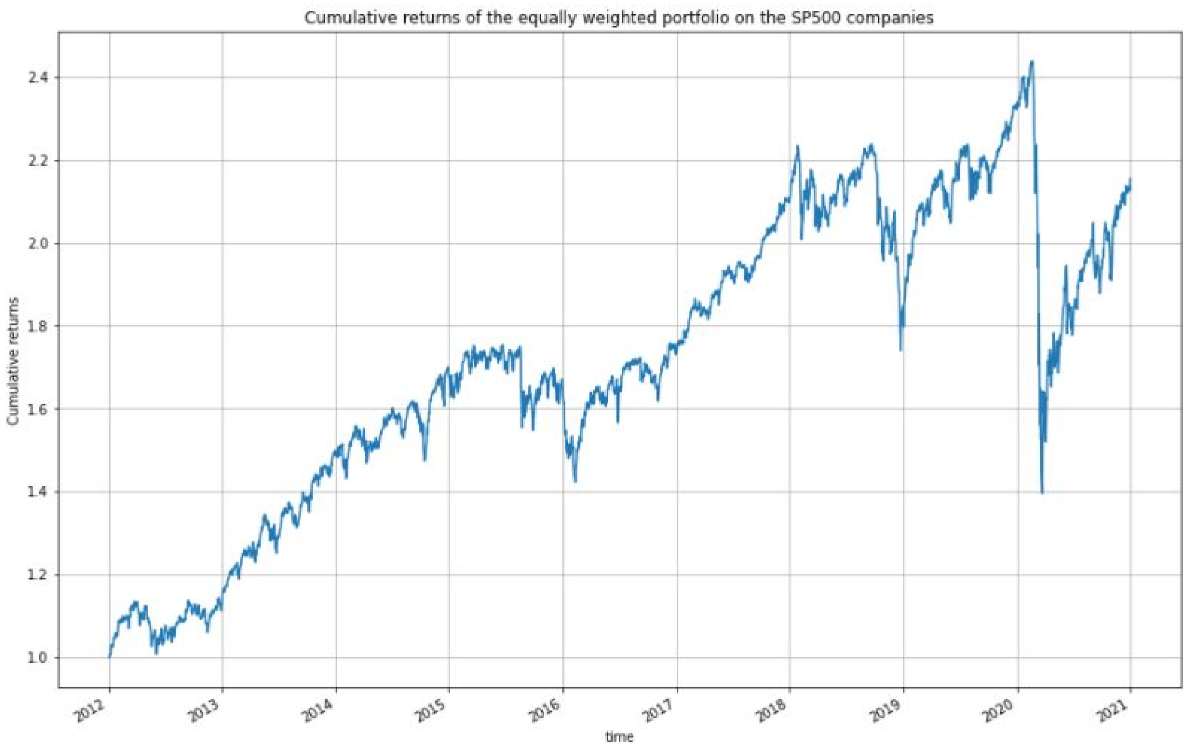


FIGURE 4.1
Cumulative Returns of the Equal Weights Portfolio

The equally weighted portfolio is here just as a baseline and reaches 2.4 before the covid crisis as a cumulative returns and bounces back to almost 2.2 afterwards.

4.2 MARKET WEIGHTS PORTFOLIO

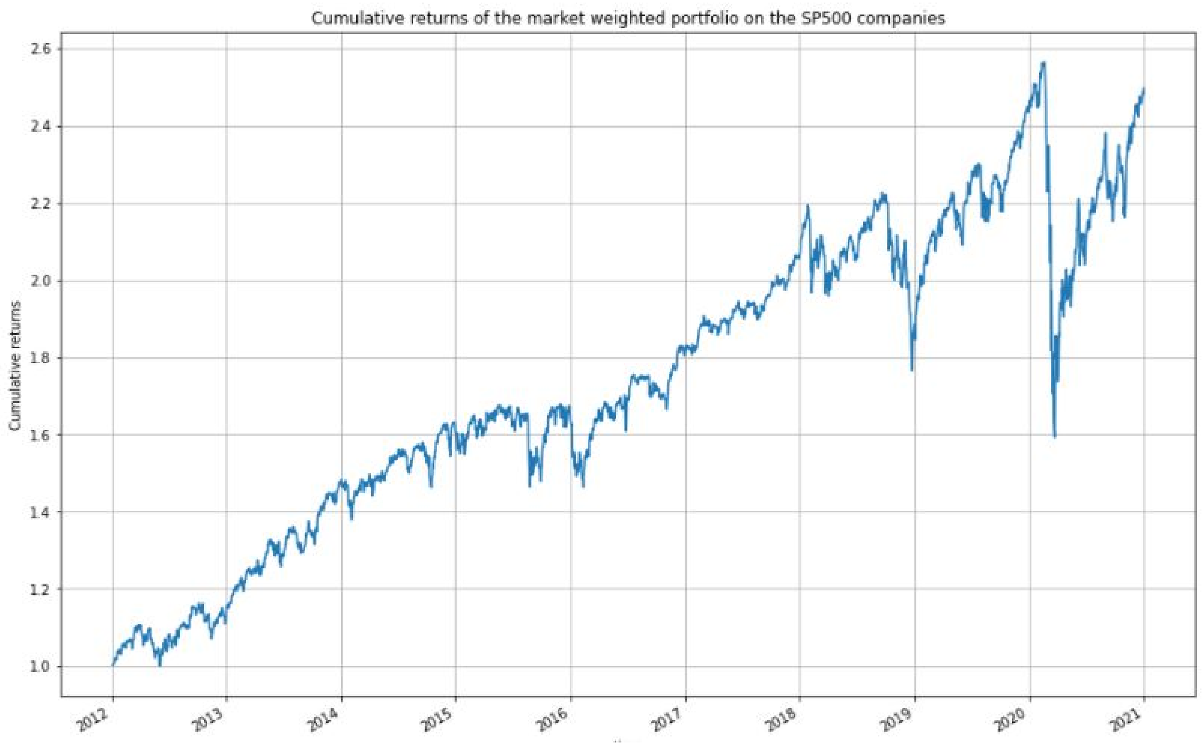


FIGURE 4.2
Cumulative Returns of the Market Weights Portfolio

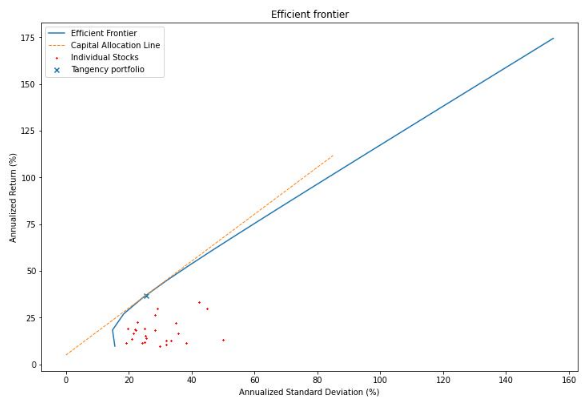
The equally weighted portfolio is here just as a baseline and reaches 2.6 before the Covid crisis and bounces back to almost 2.5 afterwards. This is known as a market proxy is a broad representation of the overall stock market. A market proxy can serve as the basis for an index fund or statistical studies. The SP 500 index is the best-known market proxy for the U.S. stock market. Index funds and exchange traded funds (ETFs) have been constructed to include all, or a portion, of the stocks in the SP 500 index. Investors and analysts use the price moves in the SP 500 as the proxy to perform various statistical research on stock market behavioral patterns. Thus, this can be seen as a CAPM estimator.

4.3 MEAN VARIANCE PORTFOLIO

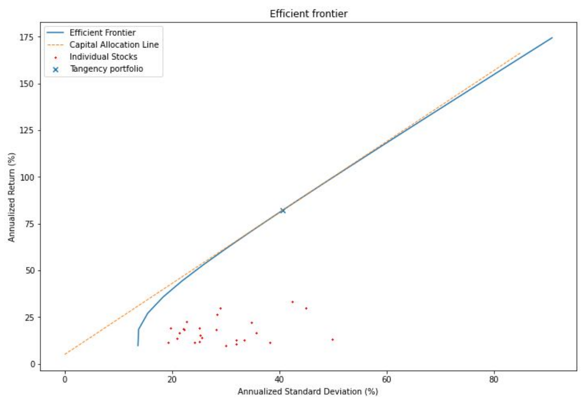
For the Mean Variance Portfolio, we first decided to analyze the performance of our portfolio using the first 25 companies in our dataset without using a rolling window. We can see the impact of the BAHC filtering. Using BAHC filtering the expected return increases significantly.

After this, we proceeded to compute our tangency portfolio using a rolling window. We chose a window of 90 days (Fig.4.4). Similarly, the most performant portfolio is the one using covariance filtering.

The cumulative returns with a window of 180 days can be found in Fig.4.5.

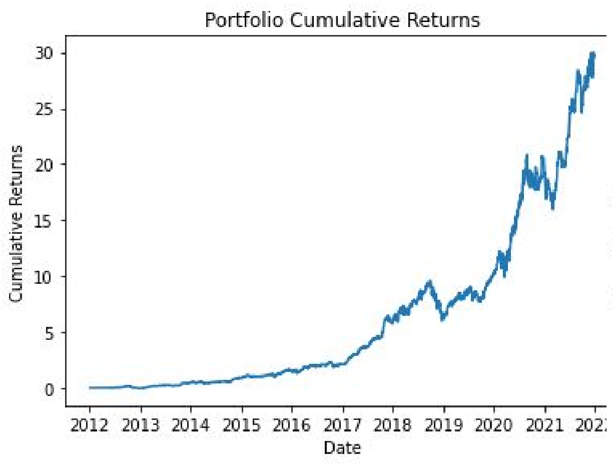


(A) Unfiltered Covariance Matrix

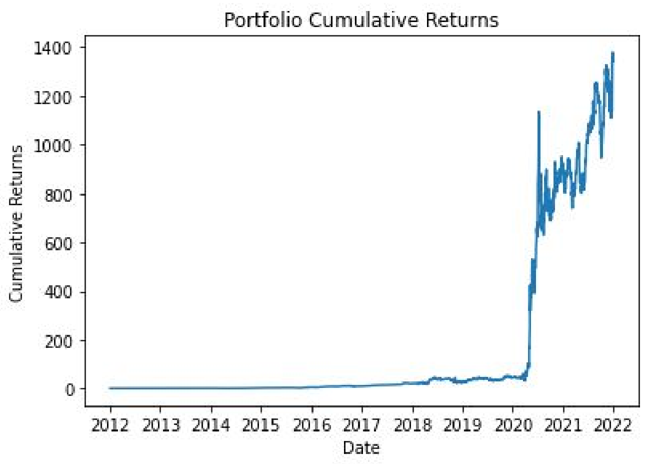


(B) BAHC-Filtered Covariance Matrix

FIGURE 4.3
Efficient Frontier for a portfolio with 25 companies

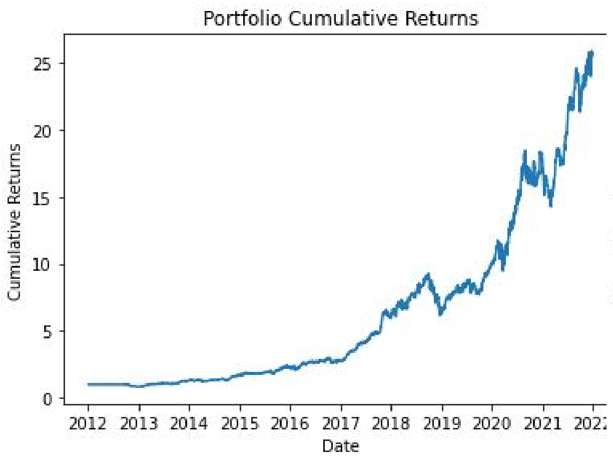


(A) Simple Covariance Matrix

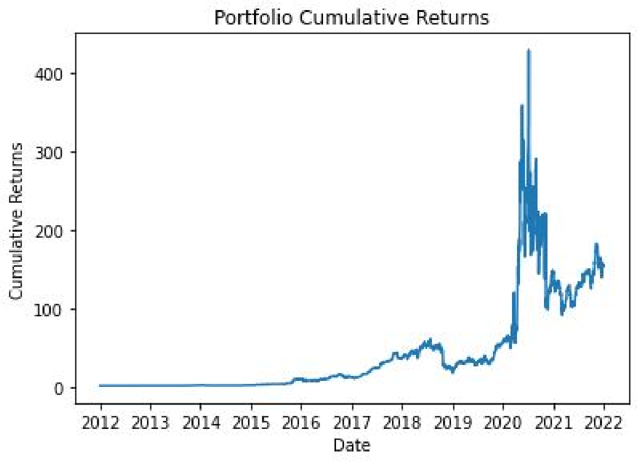


(B) BAHC-Filtered Covariance Matrix

FIGURE 4.4
Cumulative Returns of the Tangency Portfolio using a Rolling Window of 90 days



(A) Simple Covariance Matrix



(B) BAHC-Filtered Covariance Matrix

FIGURE 4.5
Cumulative Returns of the Tangency Portfolio using a Rolling Window of 180 days

We can immediately notice the impact of using BAHC. Given the fact that the Mean Variance Portfolio places more weight into better performing stocks, our model is very unstable. With both rolling windows, an explosion in returns can be seen in 2020, which is expected given the pandemic.

Nevertheless, we notice that when using a window of 180 days, our estimator is more robust. The expected returns are more reasonable, which means that our model is less sensitive to changes in our data inputs.

4.4 REINFORCEMENT LEARNING

In our case only the Actor Critic managed to converge.
In DQN finding:

$$\max_a Q\left(s', a\right)$$

turned out to be non trivial as I implemented a stochastic gradient coordinate ascent to try and find the optimal a in the continuous space of actions defined by the neural network but without any success. The code is still there and advice would be very welcomed !

Thus, this section will only present the results for the Actor Critic model. The model was trained only on $n - 100$ samples and 100 samples were left out for test.

This unlike the previous method does not allow short selling and adding short selling, and would require a bit more work to allow short selling, namely coding our loss function specialized into outputting the negative value while allowing the log-likelihood trick and certain constraint numerical methods to get values adding up to 1:

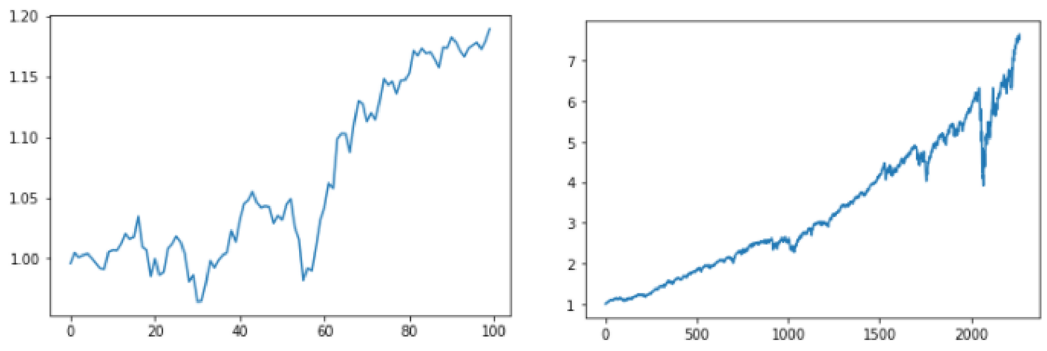


FIGURE 4.6
On the performance left test set and on the right performance the whole dataset set

Looking at the results, there a significantly better performance compared to the market portfolio. In the test set already, in the span of three months not previously seen and without borrowing, we are capable of generating a 20% return which is non negligible improvement.

The implementation of these training methods, required the use of online learning. Thus we train sample by sample which can be a bit slow. However, we heavily rely on iterators to also bypass a memory error in case the dataset cannot fit into memory. It would've been possible to speed up the training by batching, this would've also provided more stable training, but due to time constraints this could not be achieved before the deadline.

CHAPTER 5

CONCLUSION

By comparing different asset allocation models and computational techniques, we managed to get a deeper understanding of the behaviour of financial data, as well as a more structured methodology when approaching risk-management problems.

It was noticeable how by using a better estimator of the covariance matrix we were able to obtain much better results. The downside of the MVP approach is its instability. This problem could be improved by adding extra constraints on the weights. For example, adding caps on the weights, or adding a no-borrowing constraint. It was also interesting to note how the size of the rolling window affected our results. Overall, MVP is a good model since it is easy to implement and understand. By applying new techniques, like BAHC, to this classical model, we were able to obtain better performances, and understand how noisy estimators negatively affect our goal.

The reinforcement learning methods explored in the scope of this project converged to acceptable solutions for producing wealth from the market without borrowing. Althought, these methods take a long time to converge to a solution, they can very much adapt to non previously seen data very well.

Bibliography

Jiao, Wei (2003). ‘Portfolio resampling and efficiency issues’. MA thesis. Humboldt-Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät.
Zivot, Eric (2013). ‘Portfolio theory with matrix algebra’. In: *viewed on May* 5, p. 2012.
Bongiorno, Christian and Damien Challet (2021). ‘Covariance matrix filtering with bootstrapped hierarchies’. In: *PloS one* 16.1, e0245092.