Integrating Machine Learning with Modern Portfolio Theory: A Mean-Variance Approach

James Portier

University of
St Andrews

School of Mathematics and Statistics

Preface

This report, which represents 30 credits, is written during the academic year 2023-2024 as a mandatory part of my MMath (Integrated Master's) degree in Mathematics at the University of St. Andrew's. It aims at being both theoretical and practical, with the implementation being done in Python using the QuantConnect Algorithmic Trading Platform. The intended readership encompasses individuals with a foundational background in statistics, ideally complemented by knowledge of economics, machine learning, and time-series analysis. Moreover, considering the significant computational aspect of this work, readers should possess a familiarity with programming, preferably in Python. I would like to thank my supervisor Ben Swallow, a lecturer in the School of Statistics, for his instruction and guidance. I am also grateful to Campbell Wolford, a close friend of mine, for introducing me to QuantConnect last summer and providing helpful advice. Finally, I extend my deepest gratitude to my parents, Inge and Yousif; my sister, Sophie; my friends; and Adele McDaid for their love and encouragement.

I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.

James Portier.

Abstract

This dissertation explores the integration of Machine Learning (ML) techniques with Modern Portfolio Theory (MPT) for optimising investment portfolio construction through a mean-variance approach, specifically the Markowitz Mean-Variance Optimisation (MVO) model. We address the limitations of relying on historical returns by employing ML to forecast future returns accurately. The research employs the XGBoost algorithm for return direction prediction and the DCC-GARCH model for volatility forecasting, contributing to a more precise estimation of expected returns. Through the implementation on the QuantConnect Algorithmic Trading Platform using Python, we not only theoretically discuss but also practically demonstrate the superiority of the ML-enhanced MVO model over traditional methods. We find that integrating ML predictions with MPT significantly improves portfolio performance in terms of return and risk, showcasing the potential of ML in transforming financial investment strategies. The research contributes to the computational finance field by offering a novel approach to portfolio construction, combining theoretical insights with empirical analysis to highlight the benefits of merging ML predictions with traditional financial theories.

Contents

Chapter One

Introduction

1.1 Motivations and Report Outline

In the last few decades, portfolio optimisation has emerged as a challenging and interesting multi-objective problem in computational finance. In particular, using increasingly powerful computational abilities has resulted in a significant advancement in financial mathematics - the integration of Machine Learning in Quantitative Finance.

In general terms, Machine Learning (ML) encompasses the development of statistical models aimed at forecasting or determining an outcome from one or several inputs. ML is commonly used in fields such as business, medicine, and astrophysics, but it can also be used to optimise the construction of an investment portfolio. Modern Portfolio Theory (MPT), introduced by Harry Markowitz in his 1952 publication "Portfolio Selection" [21], mathematically formulates the strategy for choosing investment instruments and allocating a portion of the initial capital to each. One significant limitation of the Markowitz Mean-Variance Optimisation (MVO) model is its reliance on historical returns to estimate future performance, assuming past average returns as a proxy for future expectations. This approach often leads to inaccuracies, as examined by Black and Litterman (1991) [3]. A promising enhancement to this model involves the integration of ML techniques for forecasting future returns. By leveraging predictive analytics, ML can provide a more precise estimation of future returns, thereby potentially enhancing the overall efficacy and performance of the portfolio.

Markowitz [21] delineates the portfolio selection process into two distinct phases: initially, it necessitates the examination of past data to forecast the future trends of assets, followed by utilising these predictions to assemble the portfolio. This study endeavors to bridge

theory with application by integrating ML for predicting stock market trends during the portfolio's initial construction phase, subsequently applying these predictions to estimate expected returns. The performance of a portfolio constructed using historical returns is then evaluated against one that incorporates ML predictions.

We first start in Chapter 2 by introducing the theoretical foundations underpinning our predictive models, starting with an exploration of Machine Learning principles, then progressing to Decision Trees and the XGBoost algorithm. This is followed by an in-depth look into the GARCH model and its extension to the DCC-GARCH model for multivariate time-series data. Chapter 3 delves into Forecasting Expected Return, starting with the prediction of return direction using XGBoost, advancing to volatility forecasting with DCC-GARCH, and culminating in the computation of expected returns. Chapter 4 revisits MPT, detailing the process of portfolio construction and the nuances of MVO. The practical application of these theories is demonstrated in Chapter 5 through Backtesting, where we implement the Prediction Enabled MVO model using QuantConnect's Backtesting Environment. This section showcases the practical implementation and compares the performance of the ML-enhanced MVO model against traditional portfolio construction methods in terms of return and risk. This comprehensive approach ensures a balanced blend of theory and practical application, highlighting the efficacy of integrating Machine Learning with MPT for portfolio optimisation.

1.2 Literature Review

The predictability of stock market prices has been a subject of considerable debate, particularly in light of the Efficient Market Hypothesis (EMH). The EMH, introduced by Eugene F. Fama (1991) [9], suggests that stock prices reflect all available information, making it challenging to achieve returns that exceed average market returns on a risk-adjusted basis. Consequently, this hypothesis implies that using fundamental or historical analysis to predict future stock market behavior may be ineffective, as all known information is already incorporated in stock prices. Yet since this paper was published, several capital market anomalies have been discovered that contradict the notion of market efficiency. Jacobs (2015) [18] compiles an extensive review, encompassing over a hundred instances of capital market irregularities. These irregularities fundamentally exploit predictive indicators of returns to surpass market benchmarks.

In particular, Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, have shown promise in predicting stock market returns. LSTM models are

adept at learning long-term dependencies, making them suitable for predicting stock prices, influenced by complex sets of factors and historical trends. For instance, Zhao and Chen (2021) [29] developed a hybrid deep learning model incorporating LSTM units that demonstrated effectiveness in capturing both linear and non-linear dependencies in stock time-series data, addressing the shortcomings of traditional time-series prediction models. Another paper involving LSTM models is written by Fisher and Krauss (2017) [10], which provides a comprehensive analysis on the advantage of LSTM models over both traditional statistical methods and alternative ML models. Interestingly, their research also finds that the LSTM model's profitability is very strong up until the year 2010, after which their advantage seems to have been "arbitraged away", indicating that the market efficiency improved as traders adopted similar techniques, thereby reducing the LSTM model's edge. This stems from the limited availability of "alpha" in the market, which represents the excess return of an investment relative to the market return. While numerous studies highlight the efficacy of LSTM models for forecasting stock prices, the practical replication of these findings poses substantial challenges. Due to the model's complexity and its extensive array of hyperparameters, these models are very prone to overfitting. This complexity is compounded in the volatile environment of stock market predictions, where the inherent noise vastly overshadows the signal, making it exceptionally difficult to distinguish between meaningful patterns and random fluctuations.

After evaluating many models for our stock prediction efforts, we opted for the XGBoost Algorithm, a choice significantly swayed by the insights from a notably successful study on stock prediction, entitled: "*Predicting the direction of stock market prices using tree-based classifiers*" published by Khaidem et al. in 2019 [2]. This study reinterprets the challenge of forecasting stock market movements — rather than attempting to predict exact future prices, they recast the problem into an exercise of predicting directions. Their methodology, employing tree-based classifiers, showcases impressive predictive accuracies, achieving up to 90.23% for 1-month ahead predictions with Random Forest, and up to 86.88% using XGBoost, utilising stock data up to the 3rd of February 2017, beginning from the day each company went public. Drawing inspiration from their work with XGBoost, our study adopts a similar framework, aiming to not only replicate these high levels of directional predictive accuracy but also to utilise them to enhance the MVO model with more reliable expected return forecasts.

The integration of Machine Learning (ML) techniques with MPT is an emerging area of research. Most existing studies focus on using ML for stock *pre-selection*, which involves filtering higher-quality stocks as input for Markowitz portfolio optimisation, see Wang

et al. (2019) [27] and W. Chen et al. (2020) [5]. Whilst this approach leverages ML's predictive capabilities to enhance the traditional MPT framework, it deviates from using ML directly in the portfolio optimisation process. Over recent decades, both practitioners and scholars have expanded the Mean-Variance optimisation framework to mitigate its primary shortcomings, such as its over-reliance on historical data and the impossibility of incorporating investor perspectives. The Black-Litterman model (BL), crafted in 1990 by Fisher Black and Robert Litterman at Goldman Sachs [3], stands as a notable advancement in this endeavor. This asset allocation model provides a methodical way of combining an investor's subjective views of the future performance of a risky investment asset with the views implied by the market equilibrium. Hung et al. (2023) [15] use deep learning and natural language processing methods to construct the view distribution in the BL model. In this context, they used these methods to enhance the BL model by integrating news sentiment analysis for portfolio construction. Their empirical findings revealed that the Gated Recurrent Unit (GRU) model surpassed other deep learning models in predicting stock prices, offering a more accurate reflection of future trends. Notably, their implementation of the BL model, augmented with news sentiment measured through Google's Bidirectional Encoder Representations from Transformers (BERT) and stock price predictions via the GRU model, yielded an impressive annualized return rate of 46.6%, covering a time period from January 7, 2015, to November 20, 2019.

One study that does integrate ML with Markowitz Mean-Variance directly is a dissertation produced at Imperial College London by G. Tadlaoui (2018) [26]. In his work, Tadlaoui employs Random Forest algorithms for predicting stock directions and leverages the univariate GARCH(1,1) model for forecasting volatility, combining them to estimate the expected returns over a one-month horizon. Our research shares similarities with Tadlaoui's approach in applying ML techniques to enhance portfolio management strategies. Similar to Tadlaoui's approach, our research involves a combination of return direction prediction and volatility forecasting in order to estimate the expected returns. However, our methodology diverges significantly. We instead adopt XGBoost for direction predictions and utilise a more sophisticated model for volatility forecasting, specifically the DCC-GARCH model, which is a multivariate extension of GARCH(1,1). Furthermore, our research distinguishes itself by implementing these strategies on the QuantConnect platform, a novel approach that has not been previously explored in academic literature. This not only demonstrates the practical applicability of our methods but also expands the scope of ML in portfolio optimisation by incorporating realistic simulation environments.

Chapter Two

Model Theory

In this section, we introduce the theoretical foundations and methodologies underpinning our predictive models. We begin with an exploration of machine learning principles, progressing to the introduction of Decision Trees and the XGBoost algorithm. Following this, we introduce the univariate GARCH model, laying the groundwork for its extension to encompass multivariate time-series data. Specifically, our focus shifts to the DCC-GARCH model, setting the stage for subsequenct practical applications of these theoretical concepts.

2.1 Machine Learning

Broadly defined, machine learning is a discipline that enables computers to learn from data without being explicitly programmed for specific tasks. It is the science of programming computers so they can learn from data. Most machine learning problems fall into one of two categories: *unsupervised* or *supervised*. Unsupervised machine learning deals with data that has not been labelled, and the goal is for the model to identify patterns and relationships in the data on its own. On the other hand, supervised machine learning involves instructing the algorithm using predefined-labelled examples. For instance, the spam filter in your email is an example of a supervised machine learning application, capable of identifying spam by learning from examples of emails marked as spam (for example, by users) and regular emails, often referred to as "ham". As described by Geron [11], examples that the system uses to learn are called the *training set*, and each training example is called a *training instance*. The algorithm is given two distinct datasets: one for training and another for testing. The training dataset is employed to establish a function that maps input features to their corresponding outputs. The efficacy of this function is

then gauged by evaluating its precision in predicting the output for the data in the test set.

The spam filter example falls under a type of Supervised Learning called *Classification*. Our stock return prediction task is also exactly that - will the next-month return be positive or negative? (classified as +1, -1 respectively). A commonly used algorithm for Classification is the *Decision Tree*, which involves segmenting the predictor space into a number of simple regions.

2.2 Decision Trees and the XGBoost Algorithm

2.2.1 Decision Trees

Definition 2.1. A *Decision Tree* is a "classifier expressed as a recursive partition of the instance space" [6]. The tree consists of three types of nodes (visualised in Figure 2.1):

- **Root node**: the starting point of the decision tree. It represents the feature that best splits the data based on a certain criteria (more to follow).
- **Internal nodes**: the nodes between the root and leaves. Represent decisions based on specific feature values.
- **Leaf nodes**: terminal nodes where the final decision or prediction is made. Output is a *class label*

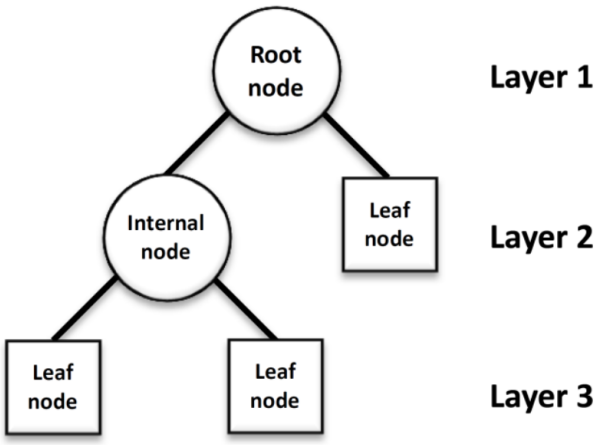

Figure 2.1: Diagram of a Decision Tree and its Nodes. Source: Hoffman [14].

In a given node, the division is achieved by selecting an attribute based on a designated *splitting criterion*. The selection of this criterion is informed by measures of impurity, such as Gini Impurity.

Definition 2.2. *Gini Impurity* serves as the metric for assessing the quality of the division within each node. Gini Impurity at node N is given by:

$$G(N) = 1 - (P_1)^2 - (P_{-1})^2 \tag{2.1}$$

where P_i is the proportion of the population with class label i.

A common intuitive strategy for selecting the optimal split at a node involves minimising impurity to the greatest extent. Thus, the most favorable split is distinguished by the maximum increase in information or the most significant decrease in impurity.

Definition 2.3. The *information gain* from executing a split is given by:

$$\text{Gain} = I(N) - p_L I(N_L) - p_R I(N_R) \tag{2.2}$$

where $I(N)$ represents the impurity measure (such as Gini or another) of node N, $I(N_L)$ is the impurity for the left child of node N following the split, and $I(N_R)$ is the impurity for the right child of node N following the split; N_L and N_R denote the left and right children of N, respectively; p_L and p_R indicate the proportions of samples in the left and right child nodes relative to the parent node.

Remark 2.4. It's important to mention that Equations (2.1) and (2.2) are applicable in scenarios where a node undergoes only two divisions. However, the specific algorithm in use might allow for multiple splits at a node, necessitating a generalisation of these equations as follows:

$$G(N) = 1 - \sum_{j=1}^{n} P_j^2 \tag{2.3}$$

$$\text{Gain} = I(N) - \sum_{j=1}^{n} p_j I(N_j) \tag{2.4}$$

where P_j is the proportion of the population with class label j, and p_j is the proportion of samples in the j-th node after the split. For more details, see the 'Supplementary Data' file provided by Khaidem et al. [2].

In Figure 2.2 we present a straightforward illustration of a decision tree applied to the Titanic dataset, utilizing Gini Impurity as the criterion for node splitting:

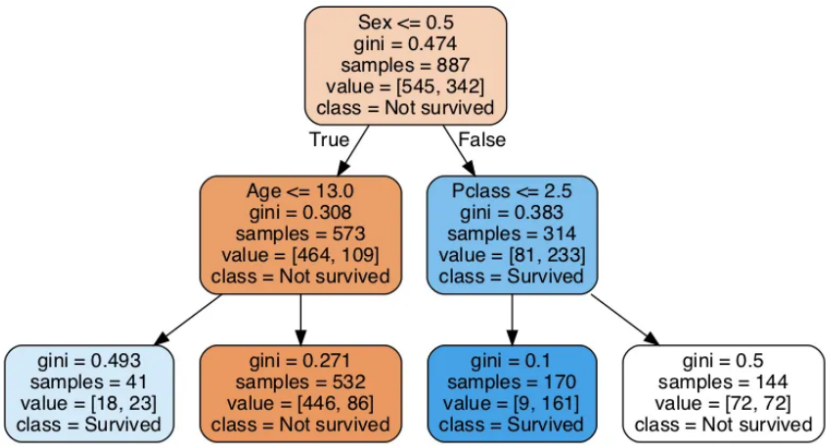

Figure 2.2: Classic example of a decision tree showing the prediction of survival for passengers on the titanic. 'Sex' is at the root node because it gives the highest information gain, as reflected by a gini impurity of 0.474 (recall how women and children were allowed to board the boats first, making gender a strong predictor of survival). Source: [7].

Although decision trees are versatile for a range of machine learning applications and offer the benefit of high interpretability, they have a significant limitation [17]. Decision trees developed to considerable depths in order to capture exceedingly complex patterns are prone to overfitting the training data, which essentially means they follow the errors, or *noise*, too closely. This scenario leads to a situation where minor fluctuations could drastically alter the tree's structure. The underlying reason for this phenomenon is their inherently low bias and elevated variance[1]. However, the predictive accuracy of trees can be significantly enhanced through the consolidation of multiple decision trees via techniques such as *bagging*, *random forests*, and *boosting*. We next discuss the theoretical framework for one of the popular boosting methods, called *XGBoost*.

[1] *Variance* here refers to the amount by which predictions would change if we estimated it using a different training data set, *bias* refers to refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. Thus a model with low bias and low variance is favourable, see *Introduction to Statistical Learning* [17].

2.2.2 eXtreme Gradient Boosting (XGBoost)

"Boosting is one of the most powerful statistical learning ideas introduced in the last twenty years" [13]. The boosting approach enables the strategic growth of trees by evaluating the effectiveness of trees previously grown within the forest. Through the process known as *additive learning*, trees are refined by incrementally integrating multiple functions. Consequently, each node is constructed in sequence, with every subsequent function aiming to more accurately address the residuals left by the preceding model. XGBoost utilises this additive methodology, whereby each new function incrementally refines an overall objective function, resulting in a cumulative assembly of functions that progressively optimises the node's approximation. Subsequently, we explore the mathematical foundations that govern XGBoost models by summarising content from the Khaidem et al. 'Supplementary Data' file [2].

Definition 2.5. The *objective function* of the XGBoost algorithm may be represented as:

$$Obj(t) = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T \tag{2.5}$$

where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the jth leaf, λ is the learning rate, $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$, and g_i and h_i are the first and second partial derivatives of the loss function respectively.

The segment within Equation (2.5) that falls under the summation is in a quadratic form. The condition for optimality in a second-order objective function is identified when its first derivative reaches zero. Therefore, inserting the value of w_j at this critical point, where the function's gradient vanishes, yields the function's optimal outcome.

$$\begin{aligned} \frac{d}{dw_j} \left[G_j w_j^* + \frac{1}{2}(H_j + \lambda)(w_j^*)^2 \right] &= 0 \\ \Rightarrow G_j + 2 \cdot \frac{1}{2}(H_j + \lambda) w_j^* &= 0 \\ \Rightarrow (H_j + \lambda) w_j^* &= -G_j \\ \Rightarrow w_j^* &= -\frac{G_j}{H_j + \lambda} \end{aligned} \tag{2.6}$$

The objective function now serves as a metric for evaluating the effectiveness of a tree. Yet, the question remains on how to ascertain the most advantageous split. Similar to

the GiniGain in Decision Trees, XGBoost utilises a measure of Gain for node division. This Gain function, akin to GiniGain, dictates that the best split at a node is the one that maximises the Gain.

$$\text{Gain} = \frac{1}{2} \cdot (\text{Gain}_L + \text{Gain}_R + \text{Gain}_{\text{Root}}) - \gamma \tag{2.7}$$

The loss function commonly used in XGBoost is the squared loss function, which is given by:

$$L = [y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))]^2 \tag{2.8}$$

For which the gradient scores are calculated as:

$$\begin{aligned} g_i &= \frac{\partial}{\partial \hat{y}_i^{(t-1)}} [y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))]^2 \\ &= 2 \cdot [y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))] \cdot [-0 - (1 + 0)] \\ &= -2[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))] \\ \Rightarrow h_i &= \frac{\partial}{\partial \hat{y}_i^{(t-1)}} \{-2[y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))]\} \\ &= -2[0 - (1 + 0)] \\ &= 2 \end{aligned} \tag{2.9}$$

The algorithm of Gradient Boosted Decision Trees, as presented in the Khaidem et al. 'Supplementary Data' file [2], is detailed by Algorithm 1 below. For a detailed pseudocode of the MaxGain(.) function below, see Appendix B.

Algorithm 1 GradientBoostedTree

input : (x_i, y_i) is the labelled training data
γ is the minimum required structure score
L is the maximum number of levels in the tree
l is the current level of the tree
λ is the learning rate
N is the number of training steps
output: A tree which is configured to predict the class label of a test sample

$l \leftarrow l + 1$;
if $l \leq L$ **then**
  $t \leftarrow 0$;
  $f \leftarrow 0$;
  while $t < N$ **do**
    estimate f_t as a regressor function, density, or distribution;
    $f \leftarrow f + f_i$;
  end while
  initialize array of scores $S[j]$;
  for $j = 1$ to n **do**
    $G_j \leftarrow 0$;
    $H_j \leftarrow 0$;
    for $i = 1$ to N **do**
      $G_j \leftarrow G_j + g_{ji}$;
      $H_j \leftarrow H_j + h_{ji}$;
      /* g_{ji} *and* h_{ji} *are the gradient statistics of the* j^{th} *sample for the* i^{th} *function* */
    end for
    $S[j] \leftarrow -0.5 \cdot G_j^2 \div (H_j + \lambda)$;
  end for
  $Cl, Cr \leftarrow \text{MaxGain}((Cl_{y_i}, Cr_{y_i}), S)$; ▷ Cl, Cr are the left and right children of this node respectively
  if Cl does not satisfy the desired level of purity **then**
    GradientBoostedTree$(Cl, \gamma, L, l, \lambda, N)$;
  end if
  if Cr does not satisfy the desired level of purity **then**
    GradientBoostedTree$(Cr, \gamma, L, l, \lambda, N)$;
  end if
end if

2.3 GARCH

2.3.1 Univariate GARCH

Historical econometric models traditionally treated volatility as a constant factor or relied on rudimentary methods for its estimation. In particular, a popular method for calculating volatility is to use the historical volatility, which assumes that the fluctuation patterns of asset prices in the past will persist into the future. To generalise this implausible assumption, a new class of stochastic processes called Auto-Regressive Conditional Heteroskedastic (ARCH) processes were introduced by Robert F. Engle (1982) [8]. These processes acknowledge the predictive power of recent events on the one-period forecast variance, reflecting the observed phenomenon of volatility clustering[2].

The Danish economist T. P. Bollerslev introduced the Generalized ARCH model, referred to as GARCH, in 1986 [4], under the mentorship of Engle during his doctoral studies. This framework extends the ARCH model, demonstrating enhanced efficacy in forecasting volatility compared to its predecessor.

Definition 2.6. The **GARCH(p,q)** model is defined as:

$$r_t = \mu_t + \epsilon_t, \tag{2.10}$$

$$\epsilon_t = \sigma_t^{1/2}\eta_t, \tag{2.11}$$

$$\sigma_t = \alpha_0 + \alpha_1\epsilon_{t-1}^2 + \cdots + \alpha_q\epsilon_{t-p}^2 + \beta_1\sigma_{t-1} + \cdots + \beta_p\sigma_{t-q}. \tag{2.12}$$

Notation:

- r_t: log return of an asset at time t.
- ϵ_t: mean-corrected return of an asset at time t.
- μ_t: the expected value of the conditional r_t.
- σ_t^2: the square of the volatility, i.e., the conditional variance at time t, conditioned on the history.
- η_t: sequence of independent and identically distributed (iid) standardized, random variables, i.e., $\mathbb{E}[\eta_t] = 0$ and $\mathrm{Var}[\eta_t] = 1$.

[2] *Volatility clustering* refers to the phenomenon of large changes in price tending to be followed by large changes, of either sign, and small changes tending to be followed by small changes.

α'_s: parameters of the model.

β'_s: parameters of the model.

p, q: order of the GARCH model.

The model's formulation clearly illustrates how the volatility at time t is influenced by past volatility levels and the historical values of the series. By incorporating parameters q for volatility clustering and p for volatility persistence, the model adeptly captures these aspects. However, a notable limitation of the GARCH model is its inability to differentiate between upward and downward market movements, attributed to the squaring of returns.

2.3.2 Multivariate GARCH

Motivation

In the realm of financial econometrics, the ability to forecast the interdependence of asset return movements holds significant importance [25]. The volatilities of financial assets tend to exhibit synchronous fluctuations to varying degrees over time and across different markets. Therefore, acknowledging and incorporating these correlated movements within a volatility model can give valuable information. Employing a multivariate approach to capture these dynamics promises to yield more pertinent models compared to the analysis through isolated univariate models. Given the predicted return direction of the XGBoost model is correct, the forecasting error of the expected return will be purely influenced by the correctness of the volatility forecast. Hence, the additional work involved in advancing the univariate GARCH model to a multivariate format is warranted for our objectives, as even marginal enhancements can notably influence the portfolio's performance.

Models of conditional variances and correlations

An important specification for Multivariate GARCH models is the conditional covariance matrix, H_t. There are many possible specifications of H_t. The Multivariate GARCH model specifications can be summarised into 4 different categories, as outlined by Silvennoinen (2007) [25]:

1. *Direct Modeling of Conditional Covariance Matrix*; This approach focuses on modeling the conditional covariance matrices, denoted as H_t, directly to capture the dynamics in variance and covariance over time.

2. *Factor-Based Models*; These models reduce dimensionality by assuming that asset returns are driven by a few unobserved common factors with GARCH properties.

3. *Conditional Variances and Correlations*; This category includes models that explicitly separate the modeling of variances and correlations, offering a more intuitive and flexible approach to understanding the dynamics of correlations.

4. *Nonparametric and Semiparametric Models*; These models offer flexibility by not strictly adhering to a specific parametric form, thus potentially providing better fit and robustness against model misspecification.

Since Python libraries for these models are limited, we opt to employ a model called *Dynamic Conditional Correlation (DCC-) GARCH* which falls within the third category mentioned above, and can be implemented using the *mgarch* library (more details to follow in Chapter 3).

2.4 DCC-GARCH

Definition 2.7. Suppose we have returns, r_t, from n assets with expected value 0 and covariance matrix H_t. Then the **Dynamic Conditional Correlation (DCC-) GARCH** model is defined as:

$$\begin{aligned} r_t &= \mu_t + \epsilon_t, \\ \epsilon_t &= H_t^{1/2}\eta_t, \\ H_t &= D_t R_t D_t. \end{aligned}$$

Notation:

- r_t: $n \times 1$ vector of log returns of n assets at time t.
- ϵ_t: $n \times 1$ vector of mean-corrected returns of n assets at time t, i.e., $\mathrm{E}[\epsilon_t] = 0$. $\mathrm{Cov}[\epsilon_t] = H_t$.
- μ_t: $n \times 1$ vector of the expected value of the conditional r_t.
- H_t: $n \times n$ matrix of conditional variances of ϵ_t at time t.
- $H_t^{1/2}$: Any $n \times n$ matrix at time t such that H_t is the conditional variance matrix of ϵ_t. $H_t^{1/2}$ may be obtained by a Cholesky factorisation[3] of H_t.

[3] A *Cholesky factorisation* is a method of decomposing a symmetric, positive-definite matrix into the product of a lower triangular matrix and its transpose, facilitating efficient numerical computations.

- D_t: $n \times n$, diagonal matrix of conditional standard deviations of ϵ_t at time t.
- R_t: $n \times n$ conditional correlation matrix of ϵ_t at time t.
- η_t: $n \times 1$ vector of iid errors such that $\mathrm{E}[\eta_t] = 0$ and $\mathrm{E}[\eta_t \eta_t'] = I$.

Comparing the DCC-GARCH specification to the univariate GARCH, we see that σ_t is replaced by H_t, which is now a matrix. As outlined in Orskaug [23], two requirements have to be considered when specifying a form of R_t:

1. The covariance matrix H_t must be *positive definite*[4]. Ensuring H_t is positive definite necessitates that R_t also be positive definite, a condition naturally met by D_t since its diagonal components are inherently positive.
2. By their nature, the values within the correlation matrix R_t must not exceed one, adhering to the mathematical definition of correlation.

Upon estimating the model's parameters, the forecast for the *conditional covariance matrix*, $H_{t+k} = D_{t+k} R_t D_{t+k}$, for a future time $t + k$ can be deduced, based on known historical data up to time t. The forecasting process for D_{t+k} and R_{t+k} regarding the covariance matrix can proceed independently. The forecast of the conditional variance is given by:

$$E[D_{t+k}|F_t] = \mathrm{diag}\left(\sqrt{E[h_{1,t+k}|F_t]}, \ldots, \sqrt{E[h_{n,t+k}|F_t]}\right)$$

where F_t denotes the information set available at time t.

In detailing the forecast for the *conditional correlation matrix*, it's important to understand that the elements within R_{t+k} do not serve as forecasts in their own right. Instead, they represent the ratio of the forecasted conditional covariance to the square root of the product of the forecasted conditional variances, specifically, $\rho_{ij} = \frac{\hat{\sigma}_{ij}}{\sqrt{\hat{\sigma}_{ii}\hat{\sigma}_{jj}}}$, where $\hat{\sigma}_{ij}, \hat{\sigma}_{ii}$, and $\hat{\sigma}_{jj}$ are the forecast elements within Q_{t+k}. Then the conditional correlatrix matrix is given by:

$$E[R_{t+k}|F_t] \approx Q_{t+k}^{*-1} Q_{t+k} Q_{t+k}^{*-1}$$

where Q_{t+k}^* is a diagonal matrix with the square root of the diagonal elements of $\hat{Q}_{t+k}$.

[4] A matrix is considered *positive definite* if it is symmetric and all its eigenvalues are positive.

Chapter Three

Expected Return Forecasting

Now that we've introduced the theory behind XGBoost and DCC-GARCH, our aim is to use these to obtain the next-month return predictions for the stocks we chose to include in our investment universe. In particular, the returns will be obtained using a combination of return direction prediction and return volatility forecasting, achieved using XGBoost and DCC-GARCH respectively. Handily, QuantConnect offers a Jupyter notebook-based "Research Environment", supporting Python along with many of its libraries. In this Environment, we can access all the data we need using the *QuantBook* class. We will use this environment to obtain the return predictions, which we will use later to backtest various strategies in Chapter 5. **All the code used to produce the results in this Chapter can be accessed from Appendix A.**

3.1 Return Direction Prediction

3.1.1 Data

QuantConnect (QC) provides a huge amount of data, with over 400TB of high quality Financial and Alternative Data readily available for analysis [24]. For our implementation, we will be using the *US Equity Coarse Universe* dataset, which is a daily universe[1] of all trading stocks in the US for a given day with the end of day price and volume. The data covers 30,000 US Equitites in total, with approximately 8,000 Equitites per day. The data starts in January 1998 and is delivered each trading day, however, we chose a timeframe starting January 1st 2009 and ending December 31st 2019. This timeframe strategically excludes the market volatility caused by the 2008 financial crisis and the

[1] In financial modeling, a *universe* refers to the total set of assets considered for trading or analysis.

economic disruptions from the 2020 COVID-19 pandemic. Thus, this period offers a stable window for analysis, reflecting a post-crisis recovery and pre-pandemic economic conditions.

3.1.2 Stock selection

Our analysis is conducted within a selected universe comprising six stocks from the S&P500, distinguished by their diverse sectors, historical volatilities, and historical returns. Furthermore, even within the same sector, these stocks exhibit a wide range of market capitalisations. The diversity of the background of the chosen companies is crucial for the generalisability of our methods. The stocks are summarised in the table below:

Company	**Stock**	**Sector**	**Historical Monthly Returns**		
			High (%)	**Low (%)**	**Avg (%)**
Ford Motor Co.	F	Automobile	97.46	-16.67	2.12
Exxon Mobil Corp.	XOM	Energy	13.39	-13.77	0.30
Devon Energy Corp.	DVN	Energy	32.37	-30.44	0.06
Microsoft Corp.	MSFT	Information Technology	21.18	-15.83	2.10
Citigroup	C	Financials	38.37	-33.33	0.66
Apple Inc.	AAPL	Information Technology	19.76	-15.53	2.25

Table 3.1: Table summarising each stock, calculated in the QC Research Environment.

In selecting stocks for our universe, we prioritised a crucial factor: diversity in performance. Rather than solely focusing on top-performing stocks, we aimed to avoid skewing our analysis. Including only high performers could obscure the effectiveness of the Machine Learning + MVO strategy compared to the equally weighted Long and Hold strategy, which would undoubtedly yield substantial returns in this scenario. To ensure a balanced evaluation, we carefully choose a range of stocks, with DVN serving as a notable example of a poorly performing stock over the chosen timeframe:

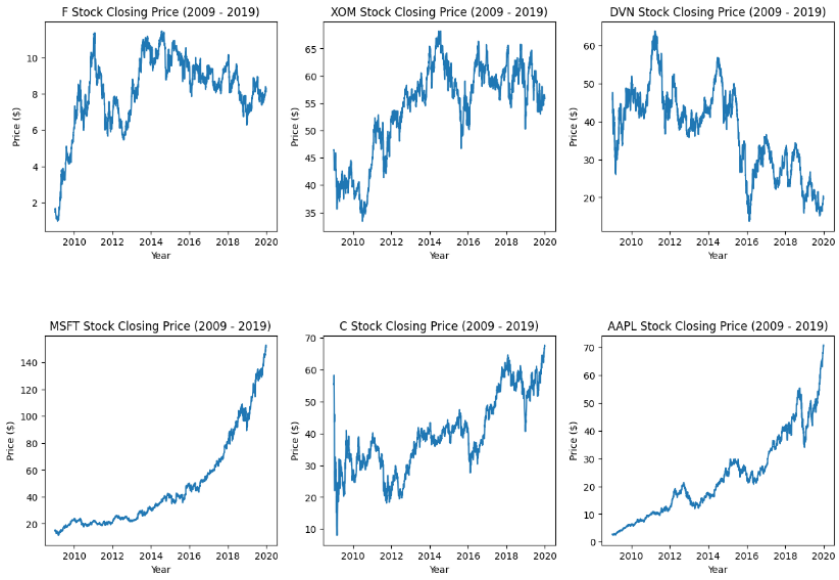

Figure 3.1: Closing price plots for each stock in our universe, plotted using the QC Research Environment.

From Figure 3.1, it is observed that the stocks in our universe exhibit varying performance and volatility over the specified timeframe.

3.1.3 Data pre-processing

QuantConnect provides exceptionally reliable data that can be accessed directly in the Research Environment, complete without any missing values. This considerably simplifies our data pre-processing efforts. One thing we do need to address however, is the fact that the data is very noisy. We employ *exponential smoothing* to mitigate the impact of sudden fluctuations and abrupt shifts in time series data. This approach achieves smoothness by assigning progressively decreasing weights to older observations, effectively averaging them out.

Definition 3.1. The *exponentially smoothed* statistic of a series Y can be recursively calculated as:

$$S_0 = Y_0$$

for $t > 0$,

$$S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1} \tag{3.1}$$

where α is the smoothing factor and $0 < \alpha < 1$. Increasing the parameter α diminishes the degree of smoothing applied. Specifically, at $\alpha = 1$, the resultant smoothed statistic

precisely matches the original observation. Such smoothing is instrumental in filtering out the random fluctuations or noise present in historical data, thereby facilitating the model's ability to discern the underlying *long-term* trends in stock price movements.

Following Khaidem et al. [2], we use $\alpha = 0.095$.

Figure 3.2: Plot of the Original vs Smoothed close prices of Ford Motor Co. Produced in the QC Research Environment.

3.1.4 Feature derivation

Input Features

Our input features consist of a variety of *technical indicators*, which are mathematical calculations based on historical prices and trading volumes that are commonly used to predict future market movements and identify trading opportunities. These indicators serve as critical inputs for the XGBoost model, aiding in the prediction of stock returns by revealing patterns and trends that may not be immediately apparent from raw data. To derive these, we require the daily Open-High-Low-Close-Volume (OHLCV) data for each of the stocks in our universe. From this input data, we easily calculate them using Python's *ta* (Technical Analysis) library[2]. The technical indicators we will use are the same as those described by Khaidem et al. [2], and are introduced subsequently.

[2]See https://technical-analysis-library-in-python.readthedocs.io/en/latest/.

Definition 3.2. The *Relative Strength Index* (RSI) is an indicator used to assess whether a stock is overbought or oversold compared to its historical price, see J. Welles [28]. It is calculated by the following formula:

$$RSI = 100 - \left(\frac{100}{1 + RS}\right) \tag{3.2}$$

where RS is the relative strength, defined as:

$$RS = \frac{\text{Average of 14 day's closes UP}}{\text{Average of 14 day's closes DOWN}}$$

The RSI spans from 0 to 100. Typically, if the RSI surpasses 70, it might suggest that the stock is overbought. Conversely, if the RSI falls below 30, it could indicate that the stock is oversold.

Definition 3.3. The *Stochastic Oscillator* (SO), introduced by Lane [19], tracks the position of a stock's closing price relative to its high and low range over a specified timeframe, typically 14 days, and tends to change before the price changes. It is given by:

$$\%K = \left(\frac{C - L14}{H14 - L14}\right) \times 100 \tag{3.3}$$

where:

- C is the most recent closing price.
- $L14$ is the lowest price traded of the 14 previous trading sessions.
- $H14$ is the highest price traded during the same 14-day period.
- $\%K$ is the current value of the stochastic oscillator.

Definition 3.4. The *Moving Average Convergence Divergence* (MACD) is a momentum and trend-following indicator that relies on the comparison of two moving averages:

$$\text{MACD} = EMA_{12} - EMA_{26}, \tag{3.4}$$

where EMA_n is the exponential moving average of the closing price over the last n days. The indicator often generates accurate entry and exit signals, with one of its notable strengths being its capability to identify the endings of significant intermediate market declines, see G. Appel [1].

Definition 3.5. The *Price Rate Of Change* (PROC), discussed by Larson and Mark, [20] is a technical indicator quantifying the percentage change between the current price and the price observed within a specified time window. It is given by:

$$ROC = \left(\frac{\text{Closing Price}_p}{\text{Closing Price}_{p-n}} \right) \times 100 \tag{3.5}$$

where:

- Closing Price_p = Closing price of most recent period
- $\text{Closing Price}_{p-n}$ = Closing price n periods before most recent period

The PROC is plotted against zero, where the indicator trends upward into positive territory when price changes are upward and dips into negative territory when price changes are downward.

Definition 3.6. The *On Balance Volume* (OBV), developed by Granville [12], leverages volume fluctuations as indicators of stock price movements. This technique aims to uncover trends in stock purchasing and selling activities by aggregating the *cumulative volume*—incrementally summing volumes on days with price increases and deducting volumes on days experiencing price declines. The OBV is given by the following piecewise function:

$$OBV(t) = OBV(t-1) + \begin{cases} \text{Volume}(t) & \text{if } P_t > P_{t-1} \\ 0 & \text{if } P_t = P_{t-1} \\ -\text{Volume}(t) & \text{if } P_t < P_{t-1} \end{cases} \tag{3.6}$$

where $P(t)$ denotes the smoothed price at time t.

Target variable

Recall our aim in this chapter is to predict the direction of the 1-month ahead return (i.e. whether a stock will have positive or negative return over the next month). Consequently, we choose our target variable to be the *sign* of the monthly log returns for each stock.

Definition 3.7. Given a vector $\mathbf{X}$ of input features and a period of 21 trading days[3], the corresponding *target variable* $Y_{21}(\mathbf{X})$ is derived as follows:

$$Y_{21}(\mathbf{X}) = \text{sign}\left(\log\left(\frac{P_{t+21}}{P_t} \right) \right)$$

where $P(t)$ denotes the smoothed close price at time t.

[3] 21 trading days correspond approximately to 1 calendar month.

Notice that we could've more easily defined the target variable by the sign of the standard returns - we chose to use the log returns here as they will be used to fit the Multivariate GARCH model, which will forecast the volatility of the returns. Crucially, this choice doesn't affect the prediction algorithm.

Date	RSI	SO	MACD	PROC	OBV	Target
2017-08-08	22.597148	6.165038	-0.072864	-5.544035	205329638.0	1
2017-09-07	22.146052	37.189935	-0.068992	-5.415028	260988848.0	1
2017-10-06	26.657916	77.307098	-0.059726	-4.586631	322945018.0	1
2017-11-04	32.424818	91.400533	-0.046467	-2.922114	365527943.0	1
2017-12-06	37.680590	90.245277	-0.031308	-2.134620	422211892.0	0

Table 3.2: Table of calculated Technical Indicators and Target Variable over the last 5 months of the training set, for Ford Motor Co.

Lets interpret the values in the first row of Table 3.2:

- The **RSI** value of 22.59 suggests that the stock may be in an oversold condition, indicating a potential buying opportunity or a forthcoming price increase if the market corrects this condition.
- An **SO** value of 6.16 is well below 20, which similarly indicates an oversold market scenario.
- The negative **MACD** value of -0.072864 indicates that the short-term momentum is weaker than the long-term momentum, suggesting a bearish[4] movement.
- A **PROC** of -5.544035% indicates that the price has decreased from the price 21 days before, signaling a downward price momentum over the past month.
- The high **OBV** value suggests significant volume on days when the price increased, potentially indicating strong buying interest in the stock. The **Target** value being 1 implies that, according to the model's classification, the price is expected to rise in the following period.

The indicators present a mixed view of the stock's condition on the date in question. While the **RSI** and **SO** suggest oversold conditions, which typically precede a reversal to

[4]"Bullish" refers to market conditions or expectations that signify an increase in stock prices, reflecting optimism or confidence in the market. Conversely, "bearish" means market conditions or expectations that signify a decrease in stock prices, reflecting pessimism or lack of confidence in the market.

positive price movement, the **MACD** and **PROC** imply bearish trends. These conflicting signals are not uncommon in technical analysis, as different indicators may reflect different aspects of market behavior and sentiment. The positive **Target** value of 1, despite the presence of both bullish and bearish signals, may indicate that the oversold conditions detected by the **RSI** and **SO** have a stronger influence on the predictive model's outcome, or it might suggest that other factors not captured by these four indicators are at play. Furthermore, the high **OBV** could be providing additional weight to the expectation of a price increase, as it reflects substantial volume on up days, potentially signaling strong buying pressure.

This complex interplay of indicators, with both bullish and bearish signals, illustrates the difficulty of manually discerning clear patterns or drawing definitive conclusions about future price movements. Such multifaceted scenarios underscore the utility of sophisticated machine learning models. In this case, we hope to use the XGBoost model to adeptly navigate through the noise and contradictions inherent in the data to identify non-linear relationships and subtle interactions between variables, providing accurate predictions for the direction of the 1-month ahead return.

3.1.5 Return direction predictions

Resampling techniques play a critical role in contemporary statistical analysis [17], involving the process of iteratively sampling from a training dataset and reapplying the model to each sample to gain deeper insights into the model's performance. A notable instance of these methods is the *Cross-Validation* technique, which entails the random partition of the dataset into two subsets: a training set for model fitting and a testing set for evaluating the model's predictive accuracy. The error rate from the test set serves as a proxy for the model's general error rate. For instance, to assess the variability of the XGBoost model's performance, distinct samples can be drawn from the training data, with the model applied anew to each sample. This procedure enables the evaluation of variations in model outcomes, providing insights that would not be apparent from a singular application of the model to the original training dataset.

Rolling window approach

For our data, the sequence and timing of the data points are significant. Traditional cross-validation techniques like *k-fold cross-validation* involve random splitting of the dataset into training and testing sets. While this method works well for datasets where

individual observations are independent of one another, it's inappropriate for time-series[5] data. In financial time-series datasets, subsequent observations are often dependent on previous ones, meaning that randomly splitting the dataset could result in using future data to predict past outcomes, which would clearly be inappropriate. Thus, we use a different approach - the Rolling Window.

The Rolling Window approach is designed to maintain the temporal order of observations. It trains the model on a continuous subset of the data (the training set) and tests it on the subsequent subset (the testing set), then rolls the window forward by a predefined step size, incrementally advancing the training and testing periods. This method ensures that predictions are always made based on information that would have been available at the time, maintaining the chronological integrity. Our rolling window implementation is designed as follows:

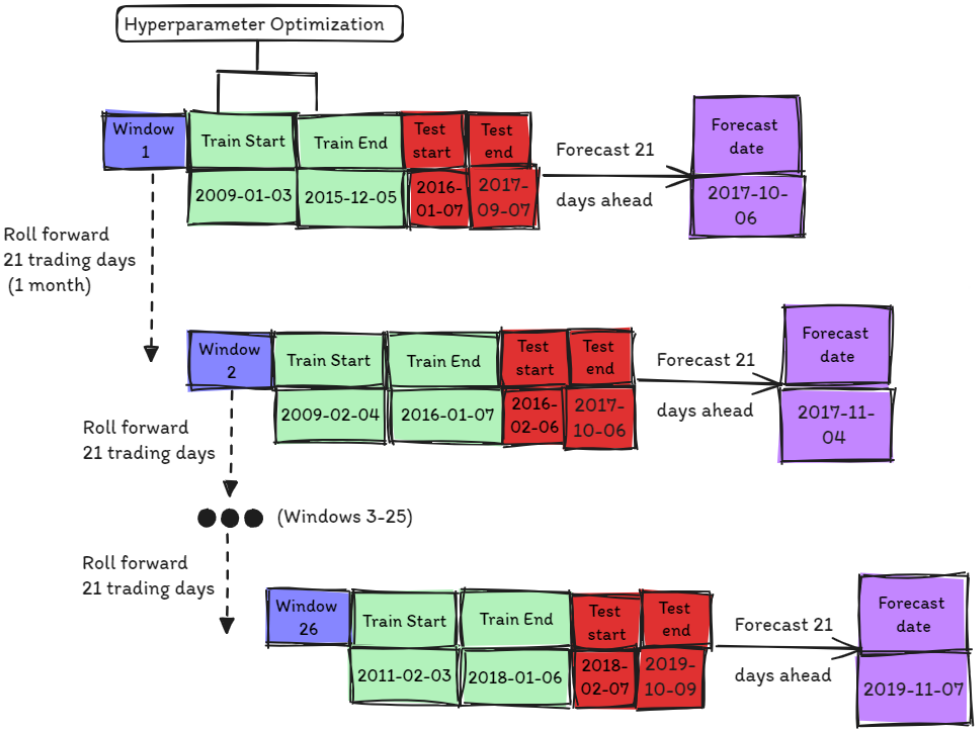

Figure 3.3: Illustration of the Rolling Window approach.

As represented on the above scheme, each window covers approximately **9 years** starting

[5]Time-series data consists of sequences of values or events obtained over successive times, often with temporal dependencies.

on 2009/01/03 (year/month/day), which is around 80% of the entire time frame. Once a window is complete, we *roll* forward by 21 trading days (approximately 1 month) after each prediction until the last train start date 2011/02/03 is reached. This gives us **26 rolling windows in total**. Each window is comprised of a train and test set, with an 80/20 train/test split, which are used to robustly assess the model's performance in each window. Additionally, the XGBoost hyperparameters are optimised in the first rolling window (more on this to follow). The whole procedure is done **once for each stock independently**, so 6 times in total.

Hyper-parameter Optimisation

The correctness of the XGBoost direction predictions are extremely important. A prediction indicating an upward trend in stock price (classified as +1) might prompt an investment or an increase in the asset's proportion within a portfolio, whereas a prediction of a downward trend suggests the opposite action. Thus, we dedicate considerable effort to fine-tuning the hyper-parameters of the XGBoost model for each stock to enhance prediction accuracy. We employ the grid-search method to systematically explore a range of hyper-parameters, seeking to optimise the balance between model complexity and generalisation to unseen data. Specifically, we focus on fine-tuning the *learning rate*, *maximum depth*, and the *number of estimators*:

- The **learning rate** regulates the size of the steps taken during each iteration as the optimisation process progresses towards minimising a loss function. A smaller learning rate may require a larger number of estimators to converge to the optimal solution, yet it often leads to better generalisation performance. We consider the range [0.01, 0.1, 0.2].
- **Max depth** limits the number of nodes in the tree. A deeper tree can model more complex patterns but also runs a higher risk of capturing noise in the training data. We consider the range [3, 5, 7].
- **N Estimators** refers to the number of boosting stages the model has to go through. More stages increase the model's capacity to learn, but too many might lead to overfitting. We consider the range [100, 200, 300].

This optimisation is executed only for the first rolling window of each stock. This is done to keep the total computational time feasible (compared to performing a grid search in each rolling window), and can be justified due to the substantial overlap in subsequent

windows — the optimal hyper-parameters identified initially can reasonably be assumed to remain effective throughout. However, it is crucial to note that the optimisation process is conducted **independently for each stock**. The distinct behavior and market dynamics of each stock justify treating them independently, allowing the optimisation process to capture each stock's unique characteristics. This leads to a set of hyper-parameters that are specifically tailored, potentially differing from stock to stock. The optimised hyper-parameters for each stock are shown in Table 3.3 below:

Stock	Learning Rate	Max Depth	N Estimators
C	0.05	3	200
MSFT	0.1	3	300
XOM	0.1	5	300
F	0.01	5	100
DVN	0.01	3	100
AAPL	0.1	7	100

Table 3.3: XGBoost Optimised Hyperparameters for each stock.

Performance measures

Note that we technically only require a training set and a single prediction (the 1-month return direction prediction) to get the monthly return directions for each stock. However, we chose to also incorporate a testing set in order to more robustly assess the quality of our predictions. We assess the reliability of the algorithm on the test set using the following metrics:

Definition 3.8. *Accuracy* measures the ratio of correct predictions (both true positives and true negatives). It is calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.7}$$

where TP, TN, FP, and FN represent the number of true positives, true negatives, false positives, and false negatives, respectively.

Whilst accuracy serves as a useful initial indicator of model performance, it possesses notable limitations, particularly in contexts with imbalanced class distributions. For instance, if 70% of the instances in a dataset are of a single class, a model could achieve a superficially high accuracy of 70% by naively predicting every instance as belonging to this majority class. Such a strategy overlooks the model's ability to accurately identify

instances of the minority class, thus inflating the perceived performance. To circumvent these pitfalls and achieve a more comprehensive evaluation, it is essential to employ additional metrics like precision, recall, and the F1 Score, which offer deeper insights into the model's predictive capabilities across different aspects [11].

Definition 3.9. *Precision* measures the accuracy of positive predictions. It is the ratio of true positive predictions to the total predicted positives:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.8}$$

This metric is particularly useful in situations where False Positives are a greater concern than False Negatives. For example, in email spam detection, a high Precision ensures that fewer non-spam emails are incorrectly marked as spam, minimising inconvenience to users.

Definition 3.10. *Recall*, also known as *sensitivity*, is the ratio of positive instances that are correctly detected by the classifier, given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.9}$$

Recall is especially important in cases where False Negatives are more critical than False Positives - in medical diagnostics, a high Recall rate is crucial to ensure that as many true cases of a disease are identified as possible, reducing the risk of missing a diagnosis.

Definition 3.11. *F1 Score* is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.10}$$

The F1 Score often provides a more informative measure than accuracy in scenarios where both types of classification errors (false positives and false negatives) are crucial to consider. For instance, in search engine ranking evaluation, where both relevance (Precision) and completeness (Recall) of the search results are equally important to user satisfaction.

Now that we've defined our approach and the metrics we will use to evaluate it, we present below the training set results for *Ford Motor Co.*:

Direction prediction results

Rolling Window	Train-test Start	Window End	Accuracy	Recall	Precision	F1
1	2009-01-03	2017-09-07	1.0000	0.9091	0.9524	0.9524
2	2009-02-04	2017-10-06	0.9091	0.7692	0.8333	0.8095
3	2009-03-06	2017-11-04	0.9091	0.7692	0.8333	0.8095
4	2009-04-04	2017-12-06	0.9167	0.7333	0.8148	0.7619
5	2009-05-06	2018-01-06	0.8182	0.7500	0.7826	0.7619
6	2009-06-05	2018-02-07	0.7273	0.8000	0.7619	0.7619
7	2009-07-07	2018-03-09	0.8333	0.7692	0.8000	0.7619
8	2009-08-05	2018-04-10	0.6923	0.7500	0.7200	0.6667
9	2009-09-03	2018-05-09	0.8462	0.9167	0.8800	0.8571
10	2009-10-03	2018-06-08	0.3846	0.6250	0.4762	0.4762
11	2009-11-03	2018-07-10	0.7692	1.0000	0.8696	0.8571
12	2009-12-03	2018-08-08	0.8333	0.8333	0.8333	0.8095
13	2010-01-05	2018-09-07	1.0000	0.8462	0.9167	0.9048
14	2010-02-04	2018-10-06	0.7500	0.6923	0.7200	0.6667
15	2010-03-06	2018-11-06	1.0000	0.7857	0.8800	0.8571
16	2010-04-07	2018-12-07	0.7273	0.8000	0.7619	0.7619
17	2010-05-06	2019-01-09	0.8182	0.6000	0.6923	0.6190
18	2010-06-05	2019-02-08	0.9167	0.8462	0.8800	0.8571
19	2010-07-07	2019-03-12	0.5833	0.8750	0.7000	0.7143
20	2010-08-05	2019-04-10	0.7692	0.8333	0.8000	0.7619
21	2010-09-03	2019-05-10	1.0000	0.9231	0.9600	0.9524
22	2010-10-05	2019-06-11	0.8333	0.8333	0.8333	0.8095
23	2010-11-03	2019-07-11	0.7273	0.8889	0.8000	0.8095
24	2010-12-03	2019-08-09	1.0000	0.8333	0.9091	0.9048
25	2011-01-04	2019-09-10	0.6667	0.8571	0.7500	0.8095
26	2011-02-03	2019-10-09	0.6000	0.6000	0.6000	0.6190

Table 3.4: Train-test results for Ford Motor Co.

The results presented in Table 3.4 are promising. The average metrics and their standard deviations—Accuracy: 0.809 ($\pm$0.150), Recall: 0.802 ($\pm$0.098), Precision: 0.798 ($\pm$0.107), and F1 Score: 0.782 ($\pm$0.109) — suggest a robust performance of the XGBoost model in forecasting stock directions across almost the entire timeframe. We now present the results for the single, end of window predictions, which are the predictions we'll actually use to forecast the expected returns.

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	1	1
2	2017-11-04	1	-1
3	2017-12-06	1	-1
4	2018-01-06	-1	-1
5	2018-02-07	-1	-1
6	2018-03-09	1	1
7	2018-04-10	1	1
8	2018-05-09	1	-1
9	2018-06-08	-1	-1
10	2018-07-10	-1	-1
11	2018-08-08	-1	-1
12	2018-09-07	-1	-1
13	2018-10-06	1	1
14	2018-11-06	-1	-1
15	2018-12-07	-1	-1
16	2019-01-09	1	1
17	2019-02-08	1	1
18	2019-03-12	1	1
19	2019-04-10	1	1
20	2019-05-10	-1	-1
21	2019-06-11	1	-1
22	2019-07-11	-1	1
23	2019-08-09	-1	-1
24	2019-09-10	-1	-1
25	2019-10-09	1	1
26	2019-11-07	1	1

Table 3.5: Actual vs XGB predicted return directions for Ford Motor Co.

There are 5 incorrect predictions during this time period (coloured in red), resulting in an overall accuracy of $21/26 = 0.808$. This closely matches the metrics measured during the training and testing phase, where we recall an overall accuracy of 0.809. It is noteworthy that the distribution of 'Up' and 'Down' directions in this period is 14/26 and 12/26, respectively. Consequently, a naive classifier—predicting exclusively 'Up' or exclusively 'Down' directions—would have achieved an accuracy of approximately 0.538 or 0.461, respectively. This highlights the superior predictive capability of our model. Detailed results for other stocks, demonstrating a similarly high level of accuracy, are presented in Appendix C.

3.2 Volatility Forecasting

In this section, we fit a Multivariate GARCH model to the log returns of each of the stocks. Using this, we forecast the volatility 1-month ahead to give us the amplitude of each stock's movement, which we will combine with the predicted directions from Section 3.1 to give us a final value for the expected return.

3.2.1 GARCH(p,q)

For comparison and confirmation of the benefit of using a Multivariate GARCH model, we also fit a univariate-GARCH(p,q) model (as introduced in Section 2.3.1). Similar to the XGBoost model fitting procedure, in each window, the GARCH(p,q) model is fitted on the on the monthly log returns from the starting date of the window date until the date at end of the test set, after which a 1-month ahead volatility forecast is made (see the rolling window outlined in Figure 3.3). Again, this is done independently for each stock. For each fitted model, we trial (p,q) values ranging from p = [1,2,3] and q = [1,2,3] using a 3x3 grid-search, keeping the combination that gives the lowest AIC[6]. In most cases however, the "best" combination turns out to be the simplest model, with p=1 and q=1.

3.2.2 DCC-GARCH(1,1)

For our DCC-GARCH implementation, we use the Python package *mgarch*[7]. We don't use a grid-search here, as this library only provides the order (1,1) specification. Recall that this model extends univariate GARCH to model multivariate time-series, allowing for the modeling of time-varying correlations between multiple assets. We therefore implement the model by **independently** fitting it on the time-series of the log returns of each of the stocks in the portoflio, along with the time-series of the log returns of a stock index they are **highly correlated** to. The indices we use for this are summarised in the table below:

[6]The *Akaike Information Criterion* (AIC) is a measure of the *relative* quality of statistical models, balancing the model's goodness of fit against its complexity to prevent overfitting [17].

[7]See https://pypi.org/project/mgarch/ for more details.

Index	Description
SPY	The SPDR S&P 500 ETF Trust, tracking the performance of the S&P 500 Index, represents the large cap U.S. equity market.
XLK	The Technology Select Sector SPDR Fund, tracking the Technology Select Sector Index, represents the technology sector.
XLE	The Energy Select Sector SPDR Fund, tracking the Energy Select Sector Index, represents the energy sector.
XLY	The Consumer Discretionary Select Sector SPDR Fund, tracking the Consumer Discretionary Select Sector Index, represents the consumer discretionary sector.

Table 3.6: Description of the selected Stock Indices.

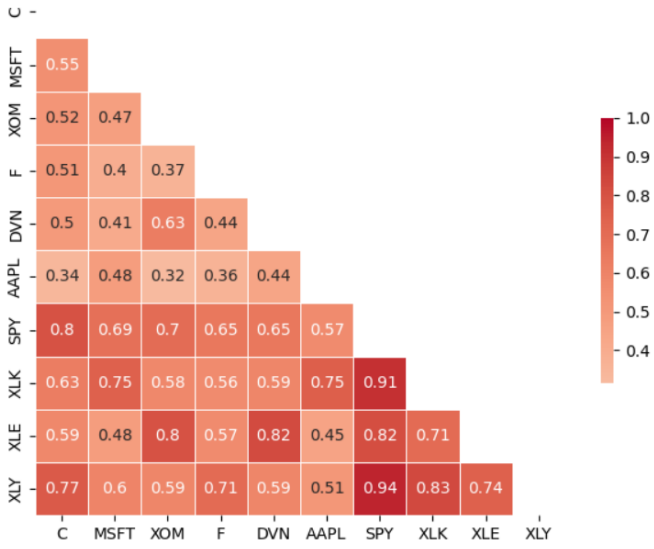

Figure 3.4: Plot of correlations between each stock and various stock indexes, calculated over the entire timeframe specified in 3.1.1. Produced in the QC Research Environment.

Figure 3.4 shows a correlation plot of each of the stocks and various indexes, calculated over the entire time frame. We see that the highest correlation between the stocks and one of the indexes is higher than the correlation between any of the stocks. This justifies the use of fitting each stock with an index instead of fitting a DCC-GARCH model on the time-series of all 6 stocks. For example, for the DVN stock (short for Devon Energy Corporation), it is the most correlated with XLE, which makes sense since XLE is an ETF that tracks the Energy Sector Index. In summary, we fit the DCC-GARCH model

independently on the monthly log-returns of each of the following stock-index pairs: (C, SPY), (MSFT, XLK), (XOM, XLE), (F, XLY), (DVN, XLE), and (AAPL, XLK).

Additionally, the *mgarch* library allows us to specify either a Multivariate Normal Distribution or Multivariate Student-t Distribution. For detailed mathematics of their specifications within the DCC-GARCH framework, see Chapter 5 of Orskaug [23]. We chose to use the latter distribution for its ability to better accommodate the outliers and heavy tails that are present in the return distributions of financial assets. Next, we compare the Univariate and the Multivariate GARCH model forecasted volatilities using two different error metrics, *MAE* and *RMSE*.

Definition 3.12. The *Mean Absolute Error* (*MAE*) is a measure of the average magnitude of errors in a set of predictions, without considering their direction, given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where n is the number of observations, y_i is the actual value, and $\hat{y}_i$ is the predicted value.

Definition 3.13. The *Root Mean Square Error* (*RMSE*) is a measure of the square root of the average of squared differences between the predicted and actual values. It is calculated as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where n is the number of observations, y_i is the actual value, and $\hat{y}_i$ is the predicted value. RMSE is sensitive to outliers and gives a higher weight to larger errors.

Stock/Index Pair	Model Used	MAE	RMSE
C	Univariate	0.03961	0.04653
C/SPY	Multivariate	0.03054	0.03597
MSFT	Univariate	0.02196	0.02634
MSFT/XLK	Multivariate	0.02060	0.02473
XOM	Univariate	0.02671	0.03011
XOM/XLE	Multivariate	0.02511	0.02983
F	Univariate	0.03085	0.04032
F/XLY	Multivariate	0.02969	0.03808
DVN	Univariate	0.06282	0.07416
DVN/XLE	Multivariate	0.05367	0.06351
AAPL	Univariate	0.03655	0.04165
AAPL/XLK	Multivariate	0.03735	0.04554

Table 3.7: Comparison of MAE and RMSE between Univariate and Multivariate GARCH Forecasts.

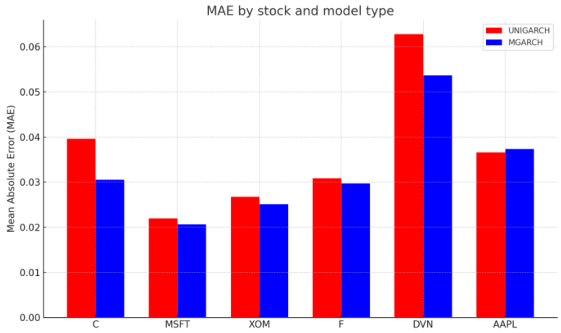

Figure 3.5: Barchart of the MAE of Univariate vs Multivariate GARCH Volatility Forecasts for each Stock.

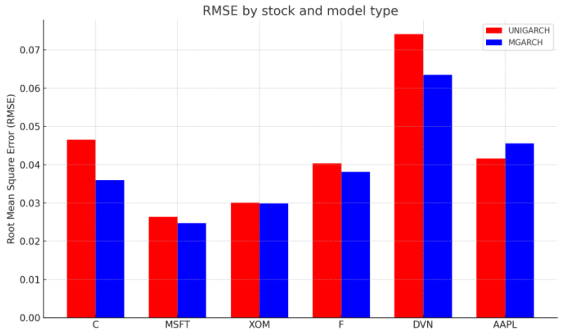

Figure 3.6: Barchart of the RMSE of Univariate vs Multivariate GARCH Volatility Forecasts for each Stock.

From Figure 3.5 and 3.6, we observe a consistent trend where the DCC-GARCH model exhibits lower Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) across most stock/index pairs compared to the Univariate GARCH model. This suggests that the DCC-GARCH model is generally more precise in its predictions, likely due to its ability to capture the dynamics in the time-varying correlation between individual stocks and their closely related indices. Table 3.7 reinforces this finding, showing numerical evidence of the superior performance of the DCC-GARCH model in terms of both MAE and RMSE for the majority of the pairs, particularly notable in the case of DVN/XLE where the reduction in error metrics is most pronounced. The only exception is with the stock AAPL, where the MAE slightly increased when the DCC-GARCH model was used. Overall, these results strongly support the use of the DCC-GARCH model to forecast the amplitude of the return, and we now have all we need to calculate the expected returns.

3.3 Obtaining the Expected Returns

Now that we have the direction predictions and the volatility forecasts for each of the stocks in our Investment Universe, we calculate the expected returns as follows:

1. For each stock and forecasting date, multiply the predicted return direction by the square root of the forecasted volatility to get the log of the expected return:

$$\texttt{log\_expected\_ret} = \texttt{np.sqrt(forecast\_volatility)} \times \texttt{xgb\_pred\_direction}$$

2. Convert the log expected return to the standard expected return as follows:

$$\texttt{expected\_ret} = \texttt{np.exp(log\_expected\_ret)} - 1$$

Note *np* refers to the NumPy package, used for scientific computing with Python. Below is a table of the forecasted returns vs the actual returns for one of the stocks:

Window	Date	Predicted Return	Actual Return
1	2017-10-06	0.047707	0.021508
2	2017-11-04	-0.044441	0.005663
3	2017-12-06	-0.052335	0.061142
4	2018-01-06	-0.048633	-0.164141
5	2018-02-07	-0.043451	-0.013941
6	2018-03-09	0.045709	0.061263
7	2018-04-10	0.056947	0.014317
8	2018-05-09	-0.047205	0.067436
9	2018-06-08	-0.051581	-0.069825
10	2018-07-10	-0.043090	-0.086532
11	2018-08-08	-0.045687	-0.064484
12	2018-09-07	-0.044805	-0.032874
13	2018-10-06	0.046342	0.063705
14	2018-11-06	-0.043873	-0.053515
15	2018-12-07	-0.052142	-0.072062
16	2019-01-09	0.055194	0.010128
17	2019-02-08	0.041633	0.036101
18	2019-03-12	0.044780	0.069686
19	2019-04-10	0.052144	0.125278
20	2019-05-10	-0.044763	-0.037255
21	2019-06-11	-0.046062	0.029532
22	2019-07-11	0.049702	-0.040288
23	2019-08-09	-0.049690	-0.002092
24	2019-09-10	-0.049641	-0.103774
25	2019-10-09	0.046386	0.060396
26	2019-11-07	0.048701	0.011211

Table 3.8: Table of the Predicted vs Actual returns for each training window for Ford Motor Co.

This is demonstrated visually in the following plots (with the plots of the forecasted expected returns for the remaining stocks given in Appendix D):

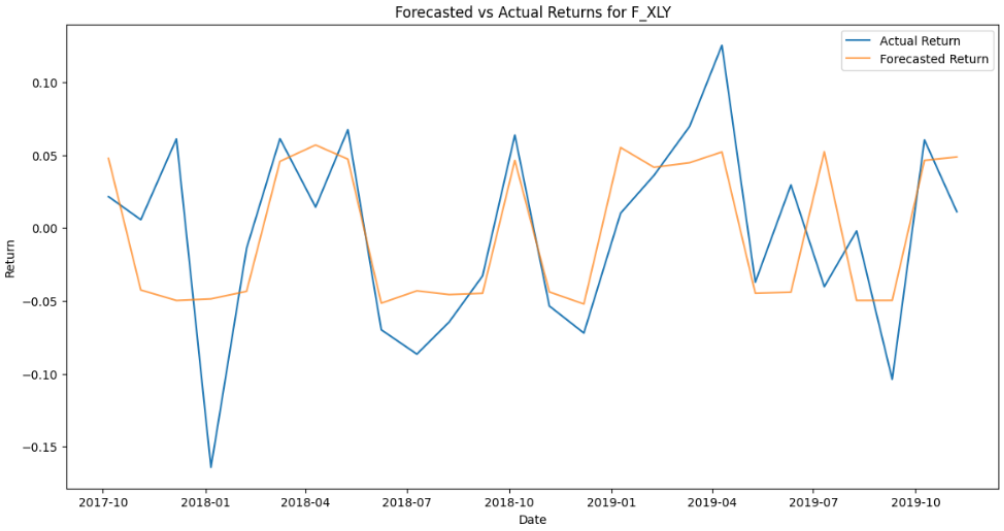

Figure 3.7: Plot of Forecasted Expected Returns for Ford Motor Co.

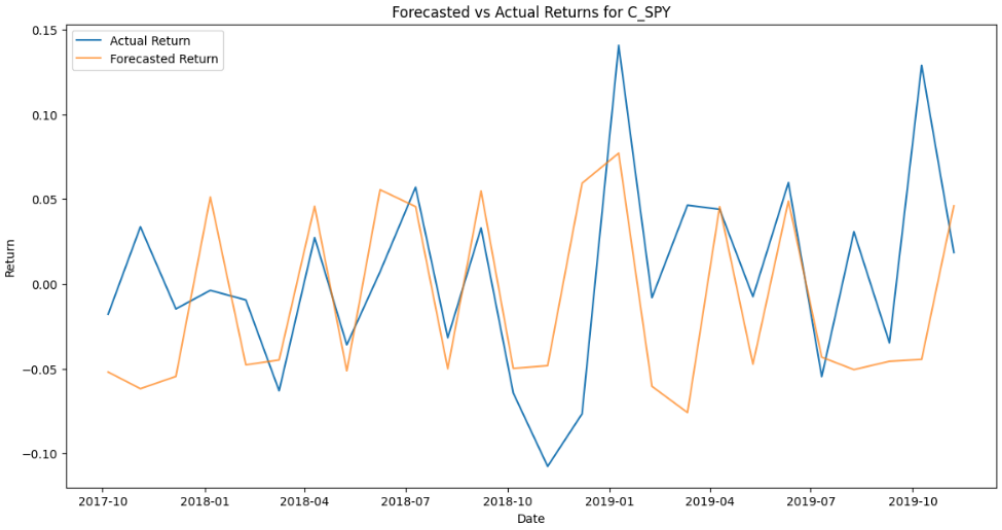

Figure 3.8: Plot of Forecasted Expected Returns for Citigroup.

When the predicted direction is incorrect, the volatility is "applied" in the wrong direction, leading to a larger discrepancy between the predicted and the actual returns. However, when the predicted direction is correct, the forecasted return closely matches the realised return in most cases, albeit with some exceptions. It appears that the DCC-GARCH model tends to underestimate volatility in instances where the realised volatility is significantly large. Such volatility spikes may be attributed to unforeseen market events, which the

model, being fitted on historical data, fails to anticipate. This limitation is inherent to the DCC-GARCH model; it is most effective in forecasting volatility when the realised volatility does not deviate excessively from the norm, either being abnormally large or small.

There is an argument to be made for advantage of the volatility forecasts being more conservative, particularly when considering inputs for Mean-Variance Optimisation (MVO). Extreme predicted values could result in disproportionate asset weights along the efficient frontier, which is undesirable. If the predicted direction is incorrect, a conservative or underestimated forecast of voltility will yield a smaller error margin compared to an overestimated forecast. This characteristic makes conservative volatility forecasts particularly beneficial for the construction of efficient portfolios.

Now that we have the predicted returns, we can use these as inputs to the Mean-Variance optimisation model and assess its performance. We first give a brief introduction to Modern Portfolio Theory.

Chapter Four

Modern Portfolio Theory

4.1 Portfolios and Mean-Variance Optimisation

A Portfolio is a collection of various financial assets, such as stocks, bonds, and cash, each assigned a specific weight. Portfolio optimisation emerges as the strategic process of optimising these weights. Investors usually tend to maximise returns and minimise risks. However, high returns typically bring increased risks. The mean–variance (MV) model proposed by Markowitz, [21] is one of the best models to solve the portfolio optimisation problem. In this model, returns and risks are quantified by means and variances, respectively, facilitating investors to make tradeoffs between maximising expected return and minimising risk. The proposed MV model lays the basis for portfolio selection. Rational investors always pursue the lowest risk under a specific expected return or the highest return under a particular risk, choosing an appropriate portfolio to maximise expected utility. The MV model aims to make a trade-off between maximising returns and minimising risks. In the following subsection, we demonstrate the theory by mathematically deriving an expression for the efficient portfolio weights, in the specific scenario where all securities are risky.

4.1.1 The Efficient Portfolio Set When All Securities Are Risky

The following is a summary of the detailed theory presented by Merton (1972) [22]. Consider a scenario involving m risky assets, where the expected return of the i^{th} asset is represented by E_i, the covariance between the returns of the i^{th} and j^{th} assets by σ_{ij}, and the variance of returns for the i^{th} asset by $\sigma_{ii} = \sigma_i^2$. Under the premise that all m assets entail risk, implying $\sigma_i^2 > 0$ for all $i = 1, \ldots, m$, and assuming no asset can be expressed

as a linear combination of the others, the returns' variance-covariance matrix, $\Omega = [\sigma_{ij}]$, is invertible. The set, or *frontier*, of all viable portfolios constructed from these m assets is characterised by the boundary of feasible portfolios, which are those with the minimum variance for a given expected return. Let x_i denote the proportion of the portfolio's value allocated to the i^{th} asset, where $i = 1, \ldots, m$, satisfying the condition $\sum_{i=1}^{m} x_i = 1$ as a fundamental constraint. Then, the frontier can be described as the set of portfolios that satisfy the constrained minimisation problem:

$$\min \frac{1}{2}\sigma^2 \tag{4.1}$$

subject to

$$\begin{aligned} \sigma^2 &= \sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j \sigma_{ij}, \\ E &= \sum_{i=1}^{m} x_i E_i, \\ 1 &= \sum_{i=1}^{m} x_i, \end{aligned}$$

where σ^2 is the variance of the portfolio on the frontier with expected return equal to E. Using Lagrange multipliers[1], (4.1) can be rewritten as:

$$\min \left[\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j \sigma_{ij}\right] + \gamma_1 \left[E - \sum_{i=1}^{m} x_i E_i\right] + \gamma_2 \left[1 - \sum_{i=1}^{m} x_i\right] \tag{4.2}$$

where γ_1 and γ_2 are the multipliers. The standard first-order conditions for a critical point are:

$$0 = \sum_{j=1}^{m} x_j \sigma_{ij} - \gamma_1 E_i - \gamma_2, \qquad i = 1, \ldots, m, \tag{4.3a}$$

$$0 = E - \sum_{i=1}^{m} x_i E_i, \tag{4.3b}$$

$$0 = 1 - \sum_{i=1}^{m} x_i. \tag{4.3c}$$

[1] *Lagrange multipliers* are a mathematical tool used for finding the local maxima and minima of a function subject to equality constraints. By introducing auxiliary variables (the multipliers) for each constraint, one can convert the constrained problem into an optimisation problem without constraints, allowing for the solution to be found by setting the gradient of this new function to zero.

Additionally, the x values that meet the three primary conditions minimise σ^2, and given the assumption regarding Ω, these solutions are unique. The system is linear with respect to the x values, thus based on (4.3a), it follows that

$$x_k = \gamma_1 \sum_{j=1}^{m} v_{kj} E_j + \gamma_2 \sum_{j=1}^{m} v_{kj}, \quad k = 1, \ldots, m, \tag{4.4}$$

where the v_{ij} are defined as the elements of the inverse of the variance-covariance matrix, i.e., $\Omega^{-1} = [v_{ij}]$. Multiplying (4.4) by E_k and summing over $k = 1, \ldots, m$, we have that

$$\sum_{k=1}^{m} x_k E_k = \gamma_1 \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_j E_k + \gamma_2 \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_k, \tag{4.5}$$

and summing (4.4) over $k = 1, \ldots, m$, we have that

$$\sum_{k=1}^{m} x_k = \gamma_1 \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_j + \gamma_2 \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj}. \tag{4.6}$$

Define:

$$A \equiv \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_j, \quad B \equiv \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_j E_k, \quad C \equiv \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj}.$$

From (4.3b), (4.3c), (4.5), and (4.6), we have a simple linear system for γ_1 and γ_2,

$$E = B\gamma_1 + A\gamma_2, \quad 1 = A\gamma_1 + C\gamma_2. \tag{4.7}$$

where we note that $\sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_j = \sum_{j=1}^{m} \sum_{k=1}^{m} v_{kj} E_k$ and that $B > 0$ and $C > 0$. Solving (4.7) for γ_1 and γ_2, we find that

$$\gamma_1 = \frac{CE - A}{D}, \quad \gamma_2 = \frac{B - AE}{D} \tag{4.8}$$

where $D = BC - A^2 > 0$. Substituting values for γ_1 and γ_2 from (4.8) into (4.4) enables the determination of the allocation ratios for each risky asset within the frontier portfolio with expected return E; specifically,

$$x_k = \frac{1}{D} \left(E \sum_{j=1}^{m} v_{kj}(CE_j - A) + \sum_{j=1}^{m} v_{kj}(B - AE_j) \right), \quad k = 1, \ldots, m. \tag{4.9}$$

Multiply (4.3a) by x_i and sum from $i = 1, \ldots, m$ to derive

$$\sum_{i=1}^{m} \sum_{j=1}^{m} x_i x_j \sigma_{ij} = \gamma_1 \sum_{i=1}^{m} x_i E_i + \gamma_2 \sum_{i=1}^{m} x_i. \tag{4.10}$$

From the definition of σ^2, (4.3b), and (4.3c), (4.10) implies

$$\sigma^2 = \gamma_1 E + \gamma_2. \tag{4.11}$$

Substituting for γ_1 and γ_2 from (4.8) into (4.11), we write the equation for the variance of a frontier portfolio as a function of its expected return, as

$$\sigma^2 = \frac{CE^2 - 2AE + B}{D}. \tag{4.12}$$

Hence, the frontier within the mean-variance domain manifests as a parabola. Analysing the first and second derivatives of (4.12) in relation to E reveals that σ^2 is a strictly convex function with respect to E, possessing a unique minimum point where

$$\begin{aligned} \frac{d\sigma^2}{dE} &= 0, \text{ i.e.,} \\ \frac{d\sigma^2}{dE} &= \frac{2}{D}(CE - A) \\ &= 0 \text{ when } E = \frac{A}{C} \\ \frac{d^2\sigma^2}{dE^2} &= \frac{2C}{D} > 0. \end{aligned} \tag{4.13}$$

Figure 4.1 shows a plot of (4.12) where $\bar{E} \equiv A/C$ and $\bar{\sigma}^2 = 1/C$ are the expected return and variance of the minimum-variance portfolio. Define $\bar{x}_k$ to be the proportion of the minimum-variance portfolio invested in the k^{th} asset, then using (4.9), we are done:

$$\boxed{\bar{x}_k = -\frac{1}{C}\sum_{j=1}^{m} v_{kj}, \quad k = 1, \ldots, m.} \tag{4.14}$$

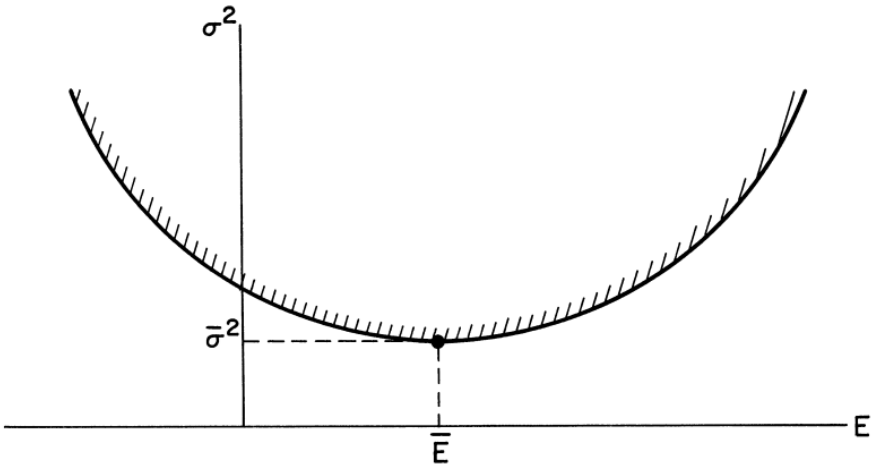

Figure 4.1: Plot of the Mean-Variance Portfolio Frontier: Risky assets only. Taken from Merton [22].

The Efficient Frontier is more commonly shown in the mean-standard deviation plane rather than the mean-variance plane. Figure 4.2 shows a clearer picture of the Efficient Frontier, with the x and y axes switched compared to Figure 4.1.

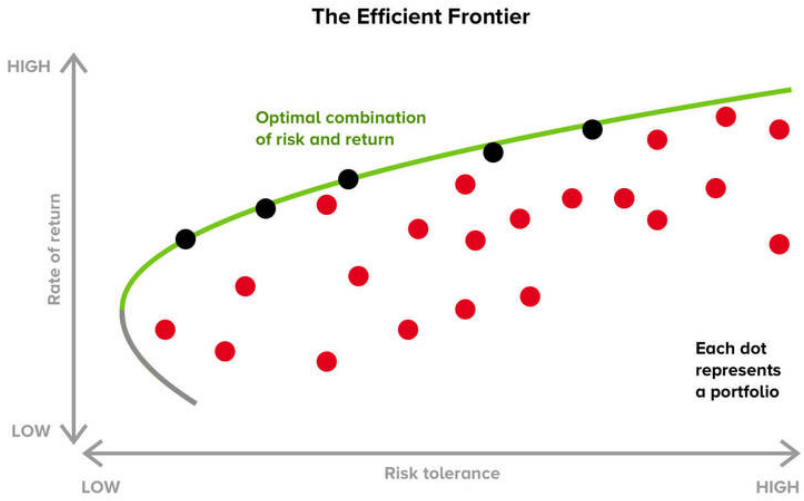

Figure 4.2: Efficient Frontier visualised. Source: [16].

The Efficient Frontier, which is the green section of the parabola-shaped curve in Figure 4.2, represents the set of portfolios that for a given level of risk, offer the highest expected

return. Markowitz Mean-Variance optimisation is a method used to determine the weights that produce the portfolio on the Efficient Frontier. Each point on the Efficient Frontier corresponds to a portfolio that is optimally constructed for the level of risk it assumes, meaning no other portfolio offers a higher return for the same level of risk. The points below the curve, coloured in red, represent portfolios that are considered sub-optimal, as they provide a lower return for the level of risk undertaken.

4.1.2 Drawbacks

In theory, Markowitz Mean-Variance optimisation works very well. In practice however, inaccuracies in the model's inputs can lead to portfolio weights that deviate from the Efficient Frontier, consequently failing to maximise returns for the assumed level of risk. In the ensuing chapter, we assess the advantages of using ML + DCC-GARCH predicted returns as part of our expected return inputs by backtesting various strategies.

Chapter Five

Backtesting

Now that we've discussed Markowitz Mean-Variance Optimisation, we can use Quantconnect's Backtesting Environment to compare MVO using the XGBoost/DCC-GARCH Forecasted Returns with using Historical Returns. From here on, we will refer to these as the "*Prediction Enabled MVO*" and "*Traditional MVO*" strategies respectively. The full backtesting results as well as the trading algorithm/research environment code are accessible in Appendix A.

5.1 Investment Strategies

5.1.1 Implementation details

QuantConnect Reality Modelling

QuantConnect's Backtesting Environment replicates the nuances of live trading, encompassing transaction costs, realistic order execution, and market liquidity constraints. It accounts for slippage, simulating the difference between expected and executed trade prices. Additionally, it considers the impact of trade size on the market and integrates potential brokerage fees, which are deducted from returns. The platform also simulates real-time market conditions using historical data with timestamps, adding a layer of authenticity by incorporating potential delays and network latencies that affect order execution. Leveraging and margin are modeled to reflect their use in real-world scenarios, including the repercussions of margin calls. Strategies can be rigorously compared to benchmarks, and various risk metrics are available to assess performance realistically. By including these complex elements, QuantConnect offers a sophisticated and holistic backtesting framework that prepares strategies for the realities of live trading, and allows

us to robustly assess whether the ML-DCC-GARCH expected returns offer an advantage over using historical returns.

Covariance inputs to MVO

Recall that in addition to the expected return of each asset in the portfolio, we also require their covariance matrix for the Mean-Variance model. The covariance matrix is crucial for MVO as it helps to understand the degree to which assets in a portfolio can offset each other's risks. Here, we operate under the assumption that volatility remains constant between two re-balancing dates, mirroring the *ex-ante* volatility calculated from the preceding month.

Using the predicted returns

In the Research Environment, the predicted returns were saved into QuantConnect's *ObjectStore*, which is a file system built into QuantConnect that allows users to save, read, and delete data, into both the Research and Backtesting Environment. In the Backtesting Environment, we read the predicted returns back into a dictionary of the predicted returns for each of our stocks. This dictionary also contains the dates corresponding to each prediction, enabling precise portfolio re-balancing on the specific dates for which the return forecasts were generated. Note that QuantConnect also facilitates the use of Machine Learning models directly in the Backtesting Environment. However, for the sake of continuity we chose not to do this as we wish to use the same predictions that were analysed in depth in Chapter 3 in the Research Environment.

Effect of drift on weights

The interval between re-balancing sessions can lead to a shift in the originally assigned portfolio weights due to variations in the closing values of assets. This shift, known as weight drift, can be problematic for an investor, particularly when it results in a few assets becoming disproportionately weighted, thereby altering the intended exposure of the portfolio. In our scenario, with re-balancing occurring on a monthly basis, the potential impact of this weight drift on our portfolio's performance is minimal and can be neglected.

5.1.2 Performance metrics

We assess our backtest outcomes through multiple performance metrics. Post-execution on QuantConnect, the platform generates comprehensive summaries relating to the strategy's performance. The selected metrics we will look at for evaluation are presented in the subsequent table.

Metric	Description	Interpretation
Return	The total gain or loss of an investment over a specific period.	A higher value indicates a more profitable investment.
Alpha	A measure of the strategy's return in excess of the market's return, as predicted by the CAPM model. Also known as the strategy's ability to beat the market (its edge).	A positive alpha suggests the strategy is outperforming the market, adjusting for the risk taken.
Sharpe Ratio	Performance compared to a risk-free asset, typically annualized.	A higher ratio indicates better risk-adjusted returns. > 1 is generally considered "good"
PSR	Probability of Superiority of Returns, indicating the likelihood that a strategy outperforms a benchmark.	A value closer to 1 suggests a higher probability of outperforming the benchmark.

Table 5.1: Description of the Backtest performance metrics, provided by QuantConnect.

5.1.3 Strategy Description

Our empirical analysis will evaluate the performance of two distinct portfolio strategies through rigorous backtesting:

1. **Long Only:** This strategy permits only the purchase (or "long" positions) of stocks within our investment universe. To adhere to the diversification prinicple of Modern Portfolio Theory, we impose weight constraints on the portfolio such that each equity's weight ranges between 0 and 0.3, with the sum of all individual stock weights being 1.

2. **Long-Short:** In this strategy, we also permit selling (or "short" positions) of stocks. By setting portfolio weight bounds between -1 and 1, the algorithm can take advantage of both positive and negative market movements, effectively increasing the potential for diversification beyond what is possible with a long-only strategy.

To ascertain the robustness and comparative performance of our investment strategies, we will execute three independent backtests for each strategy configuration. These will include: (i) Prediction Enabled MVO, (ii) Traditional MVO, and (iii) an Equal Weighted portfolio strategy that will serve as a baseline for performance benchmarking. In total, six comprehensive backtests will be conducted.

5.2 Results

5.2.1 Strategy 1 - Long only

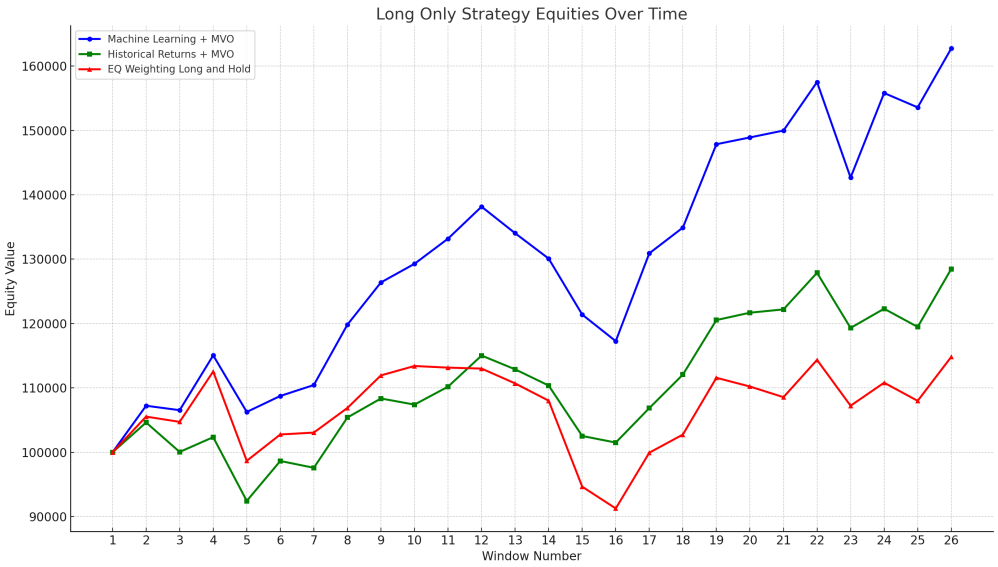

Figure 5.1: Long-only Strategy Equities over time.

Backtest	**Return**	**Alpha**	**Sharpe Ratio**	**PSR**
Prediction Enabled MVO	62.73%	0.093	0.986	52.882%
Traditional MVO	28.47%	0.005	0.457	23.336%
EQ Weighted	14.81%	-0.038	0.216	13.353%

Table 5.2: Long-Only Strategy Performance Metrics.

From Figure 5.1, it is evident that the predication enabled MVO strategy significantly surpasses the performance of the other backtests across the entire timeframe. It concludes with an equity return more than double that of the traditional MVO approach. The positive alpha of 0.093 suggests that the strategy achieved a competitive advantage over standard market returns. However, its Sharpe Ratio and PSR values are not exemplary, indicating room for improvement in risk-adjusted returns. It's important to note that the Sharpe Ratio and PSR compare the strategy's performance against the broader market, which benefits from greater diversification. This larger market context often exhibits a more stable risk-reward balance, making it a challenging benchmark. A potential improvement in the Sharpe Ratio and PSR for the Prediction Enabled MVO in the

long-only strategy would be to incorporate a larger selection of stocks. Such an approach could mitigate specific risks more effectively and exploit a broader range of opportunities, possibly leading to superior risk-adjusted performance relative to a more diversified market benchmark.

A notable observation from Figure 5.1 is the substantial decline in value experienced by all strategies between windows 12 and 16. This downturn aligns with the latter half of 2018, a notably challenging time for equities[1]. The strategies' performance during this period reflects the broader market conditions. The inherent design of the Long-Only strategy forces investment even in stocks with predicted negative one-month-ahead returns - this inflexibility is remedied by the Long-Short strategy.

We also present a bar chart in Figure 5.2 for comparative analysis of Sharpe Ratios over the entire for two investment MVO backtests. It is clear that the prediction enabled MVO backtest consistently exhibits a higher Sharpe Ratio than the traditional MVO backtest, implying that the former consistently offered better risk-adjusted returns.

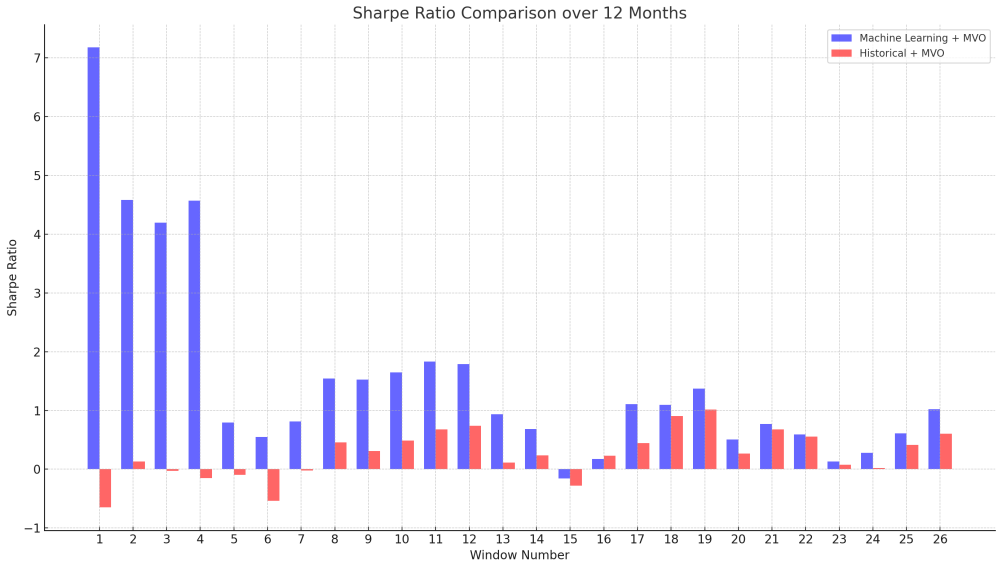

Figure 5.2: Long-only Sharpe Ratio's over time.

[1]This timeframe marked the most severe stock market downturn since the financial crisis of 2008, as detailed by https://edition.cnn.com/2018/12/31/investing/dow-stock-market-today/index.html.

5.2.2 Strategy 2 - Long-Short

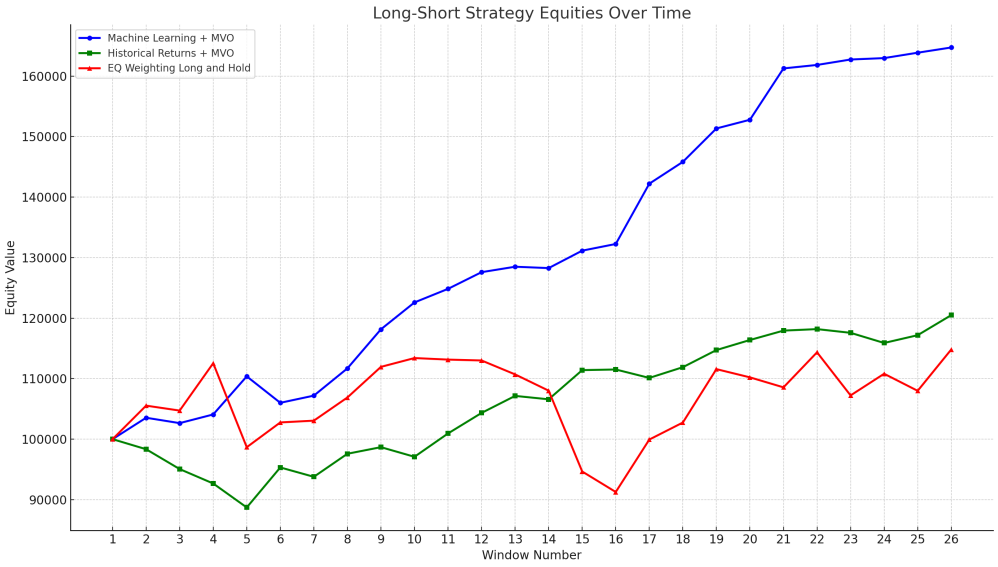

Figure 5.3: Long-Short Strategy Equities over time.

Backtest	**Return**	**Alpha**	**Sharpe Ratio**	**PSR**
Prediction Enabled MVO	64.72%	0.139	1.892	95.395%
Traditional MVO	20.49%	0.031	0.531	37.757%
EQ Weighted	14.81%	-0.038	0.216	13.353%

Table 5.3: Long-Short Strategy Performance Metrics.

The graphical representation in Figure 5.3 and the data presented in Table 5.3 display even greater performance differences between the backtests over time compared to the Long-only strategy. The Prediction Enabled MVO showcases a substantial growth in equity value, as depicted by the blue line, now outperforming the Traditional MVO by over 3x with a return of 64.72%, and an even higher alpha of 0.139.

The Sharpe Ratio of 1.892 and a PSR of 95% for the Prediction Enabled MVO strategy are now indicative of "excellent" performance. These metrics suggest not only a strong risk-adjusted return but also imply there is less than a 5% probability that the strategy failed to beat the market. Similar to the Long-Only strategy, the Prediction Enabled MVO displays a consistently higher Sharpe Ratio over the entire backtesting timeframe, as shown in Figure 5.4 below:

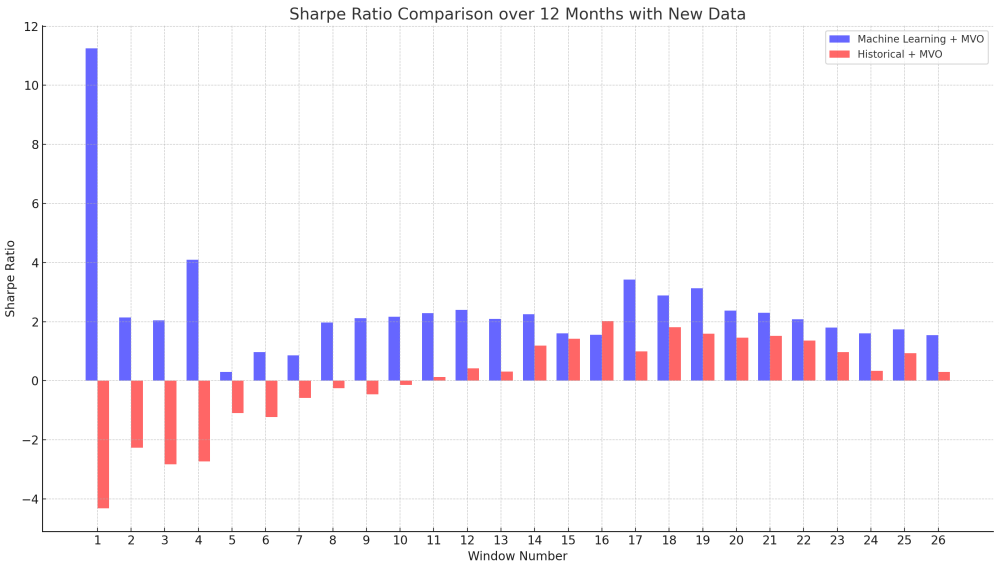

Figure 5.4: Long-Short Sharpe Ratio's over time.

Finally, the backtest for the Prediction Enabled MVO no longer exhibits the losses between windows 12 and 16, a contrast to its long-only strategy counterpart. This improvement can be attributed to the strategy's ability to take short positions, allowing it to navigate and potentially profit from downward market movements during that period. The introduction of short selling offers a strategic hedge against market downturns, which is reflected in the strategy's robust performance metrics during these windows.

Chapter Six

Conclusion

In this dissertation, we performed an in-depth exploration of integrating Machine Learning (ML) techniques with Modern Portfolio Theory (MPT) to enhance portfolio optimisation strategies. Through a combination of the XGBoost algorithm and DCC-GARCH model, this study has provided a robust framework for forecasting stock returns. These forecasts were subsequently utilised within the Mean-Variance Optimisation (MVO) framework to construct investment portfolios that outperformed those built on traditional optimisation methods.

The results certainly suggest that Machine Learning has the ability to boost the performance of an investment portfolio. Compared to the Traditional MVO, the prediction-enabled MVO exhibited +34% and +44% in absolute returns for the Long-only and Long-Short strategies respectively, which underscores the efficacy of ML-based return direction and volatility forecasts in improving portfolio performance. Additionally, the superior performance of the Prediction Enabled MVO strategy is further highlighted by its outstanding Sharpe Ratio of 1.892 and a Probability of Superiority of Returns (PSR) of 95.395% in the Long-Short strategy. These figures not only demonstrate excellent risk-adjusted returns but also imply a strong likelihood of outperforming the market. Furthermore, an Alpha of 0.139 for the Long-Short strategy signifies the strategy's exceptional capability to generate returns beyond the market's performance, considering the risk taken. This integration signifies theoretical advancements in financial mathematics and serves as a testament to the practical utility of computational power in refining portfolio management strategies, while also showcasing the superior risk-adjusted performance and market-beating potential of ML-enhanced strategies.

The dissertation contributes to the knowledge base by elucidating the practical benefits

of integrating ML with MPT in portfolio optimisation. The empirical evidence presented herein demonstrates a marked improvement in portfolio performance when ML techniques are employed to forecast future returns. Specifically, the integration of the XGBoost algorithm for return direction prediction and the DCC-GARCH model for volatility forecasting resulted in a substantial enhancement of the MVO strategy. This empirical validation of ML efficacy in financial mathematics serves to fortify the theoretical underpinnings discussed in Section 1.1, bridging the gap between theoretical potential and applied effectiveness.

The study's methodological approach, while robust, deliberately excludes periods of significant market volatility, such as the 2008 Financial Market Crisis and the onset of the Coronavirus Pandemic in 2020. This exclusion raises questions about the model's adaptability and performance under extreme market conditions. Additionally, it is likely that the advantages provided by ML techniques could be "arbitraged out" as these methods become more widespread among market participants, resulting in a narrowing of the competitive edge that such innovations initially offer.

Looking ahead, there are several pathways to enhance and expand upon the current research. Incorporating the DCC-GARCH forecasted covariance matrix as input to MVO presents an exciting opportunity to further leverage ML predictions in portfolio optimisation, potentially mirroring the improvements observed from employing ML-predicted returns over historical returns. Expanding the study to encompass a broader universe of stocks could also enhance the generalisability and applicability of the findings across various market segments. Moreover, exploring other ML models, such as Long Short-Term Memory (LSTM) networks, could offer deeper insights into capturing complex temporal dependencies in stock price movements.

In conclusion, this dissertation contributes to the growing field of computational finance by evidencing the benefits of ML and MPT integration for portfolio optimisation. While acknowledging the limitations and potential for arbitrage diminishing the ML advantages, this research lays the groundwork for future investigations. The outlined paths for further work suggest exciting opportunities for expanding the application of ML in portfolio management.

Appendices

A Access to QuantConnect Backtests and Research Environment Code

Provided below are hyperlinks that direct you to specific backtesting pages for each strategy detailed in Chapter 5. To inspect the algorithmic code used for backtesting or to review the comprehensive code developed in the research lab, simply navigate to the *Code* tab located on the backtesting page. Within this section, the trading algorithm pertinent to each discussed strategy is contained in the *main.py* file. Additionally, Jupyter Notebooks crafted within the Research Environment, encompassing any work performed in Chapter 3, are identifiable by files with the *.ipynb* extension.

Long-Only Strategies

Prediction Enabled MVO

Traditional MVO

Equal Weighted

Long-Short Strategies

Prediction Enabled MVO

Traditional MVO

Equal Weighted

B Pseudocode for the MaxGain algorithm

Taken from the Khaidem et al. 'Supplementary Data' file [2].

Algorithm 2 MaxGain

input : $(x_i, y_i)^m$ is the labelled training data
        $S[:]$ is the list of structure scores
output: Left and right child nodes, ensuring maximum purity after split

Sort $(x_i, y_i)^m$ and $S[:]$ in the increasing order of $S[:]$;
$Gain_{split} \leftarrow -\infty$ ▷ $Gain_{split}$ stores the value of the gain after each split
$Gain_N \leftarrow \sum S[:]$ ▷ the overall score of the parent node
$SplitPoint \leftarrow -1$ ▷ stores the best split point
$f \leftarrow 0$;
for $j := 1$ to $m - 1$ **do**
    $N_L \leftarrow S[0 : j]$ ▷ list of scores of left child
    $N_R \leftarrow S[j + 1 : m]$ ▷ list of scores of right child
    $G_L \leftarrow \sum N_L$ ▷ overall gain of left child
    $G_R \leftarrow \sum N_R$ ▷ overall gain of right child
    $Gain \leftarrow G_L + G_R - Gain_N$ ▷ gain calculated in the j^{th} iteration
    if $Gain > Gain_{split}$ and $Gain > \gamma$ **then**
        $Gain_{split} \leftarrow Gain$;
        $SplitPoint \leftarrow j$;
    end if
end for

return $(x_i, y_i)^j, (x_i, y_i)^{j+1}$;

C Direction Prediction Results

XOM Direction Predictions

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	1	1
2	2017-11-04	1	1
3	2017-12-06	1	1
4	2018-01-06	-1	1
5	2018-02-07	-1	-1
6	2018-03-09	1	1
7	2018-04-10	1	1
8	2018-05-09	1	1
9	2018-06-08	1	1
10	2018-07-10	-1	-1
11	2018-08-08	1	1
12	2018-09-07	1	-1
13	2018-10-06	-1	-1
14	2018-11-06	-1	-1
15	2018-12-07	-1	-1
16	2019-01-09	1	-1
17	2019-02-08	1	1
18	2019-03-12	1	-1
19	2019-04-10	-1	-1
20	2019-05-10	-1	-1
21	2019-06-11	1	1
22	2019-07-11	-1	-1
23	2019-08-09	-1	-1
24	2019-09-10	-1	-1
25	2019-10-09	1	1
26	2019-11-07	-1	-1

Table 6.1: Actual vs XGB predicted return directions for Exxon Mobil Corp.

DVN Direction Predictions

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	1	-1
2	2017-11-04	-1	-1
3	2017-12-06	1	1
4	2018-01-06	-1	-1
5	2018-02-07	-1	1
6	2018-03-09	-1	-1
7	2018-04-10	1	1
8	2018-05-09	1	1
9	2018-06-08	1	1
10	2018-07-10	-1	1
11	2018-08-08	-1	-1
12	2018-09-07	-1	-1
13	2018-10-06	-1	1
14	2018-11-06	-1	-1
15	2018-12-07	-1	-1
16	2019-01-09	-1	-1
17	2019-02-08	1	1
18	2019-03-12	1	-1
19	2019-04-10	1	1
20	2019-05-10	-1	-1
21	2019-06-11	1	-1
22	2019-07-11	-1	-1
23	2019-08-09	1	1
24	2019-09-10	-1	-1
25	2019-10-09	1	-1
26	2019-11-07	1	1

Table 6.2: Actual vs XGB predicted return directions for Devon Energy Corp.

MSFT Direction Predictions

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	1	1
2	2017-11-04	-1	1
3	2017-12-06	1	1
4	2018-01-06	1	1
5	2018-02-07	1	1
6	2018-03-09	-1	1
7	2018-04-10	1	1
8	2018-05-09	1	1
9	2018-06-08	1	1
10	2018-07-10	1	-1
11	2018-08-08	1	1
12	2018-09-07	1	1
13	2018-10-06	-1	-1
14	2018-11-06	1	1
15	2018-12-07	-1	-1
16	2019-01-09	1	-1
17	2019-02-08	1	1
18	2019-03-12	1	-1
19	2019-04-10	1	-1
20	2019-05-10	1	1
21	2019-06-11	1	1
22	2019-07-11	1	1
23	2019-08-09	-1	-1
24	2019-09-10	-1	-1
25	2019-10-09	1	1
26	2019-11-07	1	1

Table 6.3: Actual vs XGB predicted return directions for Microsoft Corp.

C Direction Predictions

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	-1	-1
2	2017-11-04	1	-1
3	2017-12-06	-1	-1
4	2018-01-06	-1	1
5	2018-02-07	-1	-1
6	2018-03-09	-1	-1
7	2018-04-10	1	1
8	2018-05-09	-1	-1
9	2018-06-08	1	1
10	2018-07-10	1	1
11	2018-08-08	-1	-1
12	2018-09-07	1	1
13	2018-10-06	-1	-1
14	2018-11-06	-1	-1
15	2018-12-07	-1	1
16	2019-01-09	1	1
17	2019-02-08	-1	-1
18	2019-03-12	1	-1
19	2019-04-10	1	1
20	2019-05-10	-1	-1
21	2019-06-11	1	1
22	2019-07-11	-1	-1
23	2019-08-09	1	-1
24	2019-09-10	-1	-1
25	2019-10-09	1	-1
26	2019-11-07	1	1

Table 6.4: Actual vs XGB predicted return directions for Citigroup.

AAPL Direction Predictions

Window	Forecast Date	Actual Direction	XGB Predicted Direction
1	2017-10-06	1	1
2	2017-11-04	-1	-1
3	2017-12-06	1	1
4	2018-01-06	-1	-1
5	2018-02-07	1	-1
6	2018-03-09	-1	-1
7	2018-04-10	1	1
8	2018-05-09	1	1
9	2018-06-08	-1	-1
10	2018-07-10	1	1
11	2018-08-08	1	1
12	2018-09-07	1	1
13	2018-10-06	-1	1
14	2018-11-06	-1	-1
15	2018-12-07	-1	-1
16	2019-01-09	1	1
17	2019-02-08	1	1
18	2019-03-12	1	1
19	2019-04-10	1	1
20	2019-05-10	-1	-1
21	2019-06-11	1	1
22	2019-07-11	1	-1
23	2019-08-09	1	1
24	2019-09-10	1	-1
25	2019-10-09	1	1
26	2019-11-07	1	1

Table 6.5: Actual vs XGB predicted return directions for Apple Inc.

D Expected Return Forecasts

XOM Return Forecasts

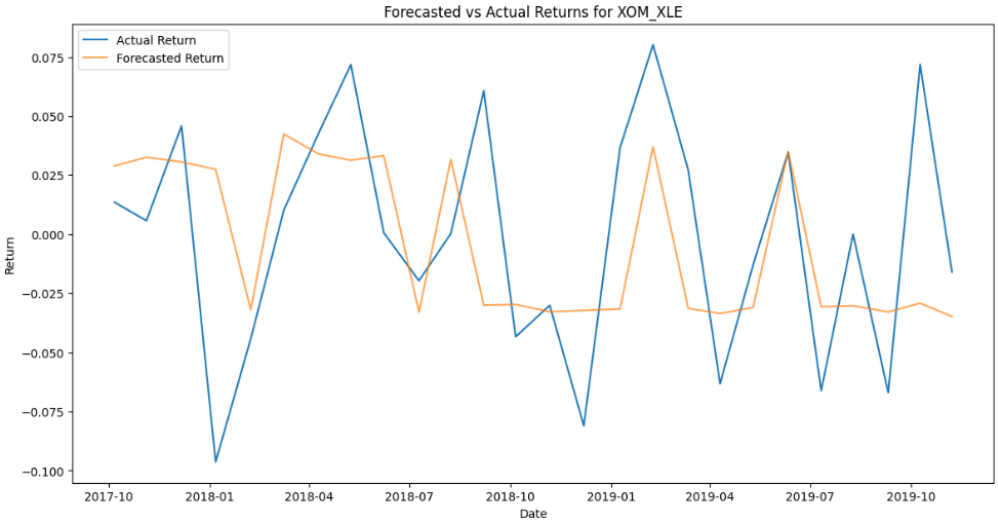

Figure 6.1: Plot of Forecasted Expected Returns for Exxon Mobil Corp.

DVN Return Forecasts

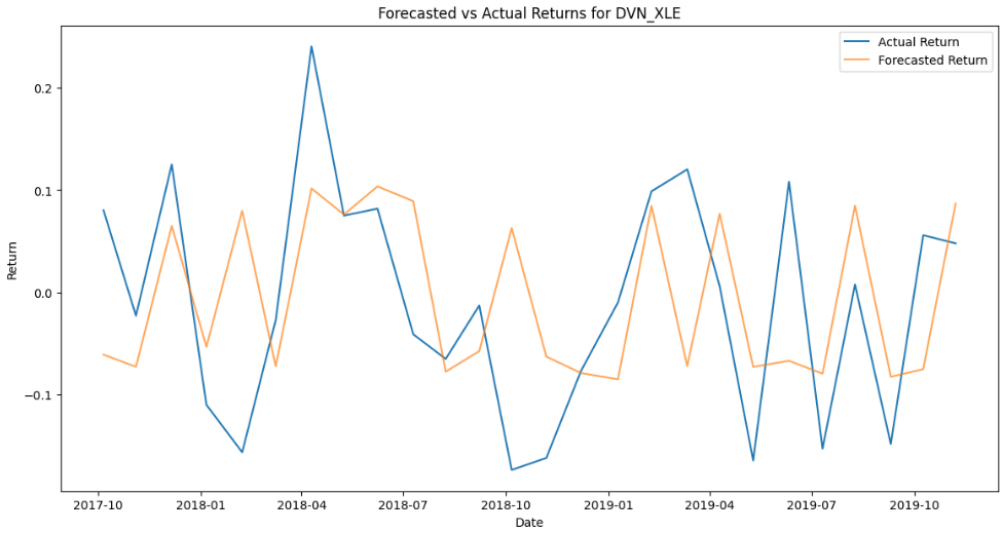

Figure 6.2: Plot of Forecasted Expected Returns for Devon Energy Corp.

MSFT Return Forecasts

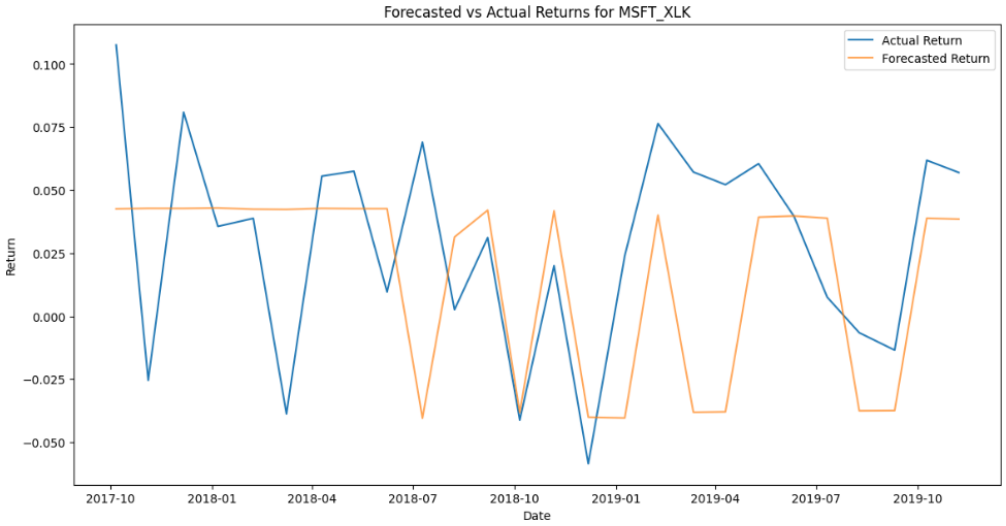

Figure 6.3: Plot of Forecasted Expected Returns for Microsoft Corp.

AAPL Return Forecasts

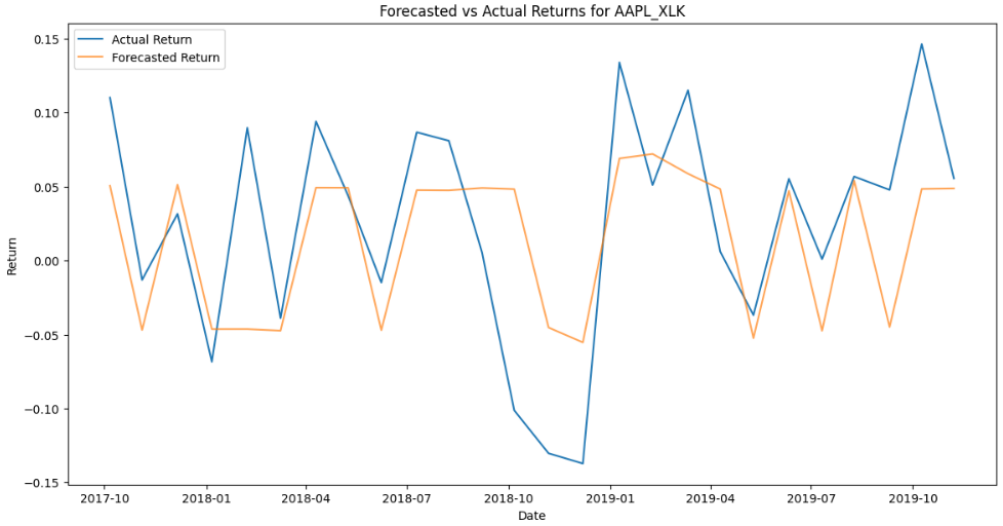

Figure 6.4: Plot of Forecasted Expected Returns for Apple Inc.

Bibliography

[1] Appel, G. (2005). *Technical Analysis Power Tools for Active Investors.*

[2] S.Basak, Khaidemn L., Saha S., Kar S. (2019). *Predicting the direction of stock market prices using tree-based classifiers.* The North American Journal of Economics and Finance, Vol. 47.

[3] Black, F., & Litterman, R. (1990). *Asset allocation: combining investor views with market equilibrium.* Goldman Sachs Fixed Income Research.

[4] T. Bollerslev. (1986). *Generalized autoregressiv conditional heteroskedasticity.* Journal of econometrics, Vol. 31.

[5] Wei Chen, Haoyu Zhang, Mukesh Kumar Mehlawat, Lifen Jia. (2020). *Mean–variance portfolio optimisation using machine learning-based stock price prediction.* Applied Soft Computing, Vol. 100.

[6] Dahan, H., Cohen, S., Rokach, L., Maimon (2014). *Proactive Data Mining with Decision Trees.*

[7] Duif, M. (2020). *Titanic Decision Tree example.* Link: https://towardsdatascience.com/an-introduction-to-decision-trees-with-python-and-scikit-learn-1a5ba6fc204f.

[8] Robert F. Engle. (1982). *Autoregressive Conditional Heteroscedasticity with estimates of the Variance of United Kingdom inflation.* Econometrica, Vol. 50, No. 4.

[9] Eugene F. Fama. (December 1991). *Efficient Capital Markets: II.* The Journal Of Finance, Vol XLVI, No 5.

[10] Thomas Fischer, Christopher Krauss. (2017). *Deep learning with long short-term memory networks for financial market predictions.* European Journal of Operational Research, V. 270.

[11] Aurélien Géron. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

[12] Granville, J.E. (1976). *Granville's New Strategy of Daily Stock Market Timing for Maximum Profit*.

[13] T. Hastie, R. Tibshirani, J. Friedman. (2001). *The Elements of Statistical Learning*.

[14] Hoffman, K. (2020). *Decision Tree and its Nodes*. Link: https://ken-hoffman.medium.com/decision-tree-hyperparameters-explained-49158ee1268e.

[15] Ming-Ching Hung et al. (2023). *Intelligent portfolio construction via news sentiment analysis* International Review of Economics & Finance, Vol. 89.

[16] Inyova. (2019). *Effcient Frontier Diagram*. Link: https://inyova.ch/en/expertise/efficient-frontier-investment-theory/.

[17] Gareth James, Witten, Hasti, Tibshirani (2013) *An Introduction to Statistical Learning: with Applications in R*.

[18] H. Jacobs, (2015). *What explains the dynamics of 100 anomalies?* Journal of Banking Finance, Vol. 57.

[19] Lane, G. (1984). *Lanes stochastics*. Second issue of Technical Analysis of Stocks and Commodities magazine, Vol. 2:3.

[20] Larson, Mark (Ed.). (2015). *Price rate of change: 12 simple technical indicator that really work*.

[21] H.M Markowitz. (1952). *Portfolio selection*. The Journal of Finance, Vol. 7, No. 1.

[22] Robert C. Merton (1972). *An Analytic Derivation of the Efficient Portfolio Frontier*. The Journal of Financial and Quantitative Analysis, Vol. 7, No. 4.

[23] Elisabeth Orskaug. (2009). *Multivariate DCC-GARCH Model -With Various Error Distributions*. Norwegian University of Science and Technology.

[24] *QuantConnect Data*. Link: https://www.quantconnect.com/datasets

[25] A. Silvennoinen, T. Terasvirta. (2007). *Multivariate garch models*. Working Paper Series in Economics and Finance, No. 669.

[26] Ghali Tadlaoui (2018) *Intelligent Portfolio Construction: Machine-Learning enabled Mean-Variance Optimization*. Department of Mathematics, Imperial College London.

[27] Wang, Li, Zhanga, Liu. (2019). *Portfolio formation with preselection using deep learning from long-term financial data*. Expert Systems with Applications, Vol. 143.

[28] Wilder, J. Welles (1978). *New concepts in technical trading systems.*

[29] Yang Zhao, Zhonglu Chen. (2021). *Forecasting stock price movement: new evidence from a novel hybrid deep learning model*. Journal of Asian Business and Economic Studies, Vol. 25.