Summer 2024

Comparative Study of Crypto Volatility and Price Forecasting using a Mixture of Time Series and Machine Learning Models

Abhishek Kafle
Missouri University of Science and Technology

Department:

COMPARATIVE STUDY OF CRYPTO VOLATILITY AND PRICE FORECASTING USING A MIXTURE OF TIME SERIES AND MACHINE LEARNING MODELS

by

ABHISHEK KAFLE

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

APPLIED MATHEMATICS

2024

Approved by:

Akim Adekpedjou, Advisor
Martin Bohner
V.A. Samaranayake

ABSTRACT

Forecasting financial product volatility and price is crucial for informed decision-making in investment and risk management. The models considered include GARCH, LSTM, GRU, BiLSTM, and hybrid models that incorporate various combinations of these models. We present a comparative analysis of forecasting volatility and price using the aforementioned models.

We also introduce a user-friendly dashboard for model training and evaluation, enabling users to upload datasets and customize model parameters. The dashboard allows users to select the type of model, specify the dataset range for training, determine the number of epochs, adjust the number of layers for deep learning methods, and set the window size for data processing. After selection of parameters, models are trained in the backend and the dashboard presents comprehensive results that includes graphical representations and performance metrics. These metrics facilitate comparison between the models in terms of accuracy, robustness, and computational efficiency.

Through empirical analysis, we demonstrate the effectiveness of different models in forecasting financial product volatility and price. Our findings provide insights into the strengths and limitations of single and hybrid models.

ACKNOWLEDGMENTS

First and foremost, I am deeply thankful to my advisor, Dr. Akim Adekpedjou, for his invaluable guidance, support, and encouragement throughout the research process. His expertise, patience, and mentorship have been instrumental in shaping this work. I am also grateful to the members of my thesis committee, Dr. Martin Bohner and Dr. V.A Samaranayake, for their valuable insights, feedback, and suggestions, which have greatly enriched the quality of this thesis. I would also like to extend my appreciation to the department of Mathematics at Missouri S&T for providing the necessary resources and facilities for this project.

Special thanks to my family and friends for their unwavering support, understanding, and encouragement during this journey. Their love, encouragement, and belief in me have been a constant source of strength and motivation. I am indebted to all the participants and individuals who generously shared their time, knowledge, and experiences, without whom this research would not have been possible.

Lastly, I would like to acknowledge the countless researchers, scholars, and authors whose work has inspired and informed this thesis. Thank you all for your invaluable contributions.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

1. INTRODUCTION

1.1. BACKGROUND

Financial volatility forecasting is a critical component of risk management and investment decision-making in financial markets [1]. Volatility is defined as the degree of variation of returns for a given financial asset over a specific period of time, and plays a crucial role in determining market dynamics and asset pricing [2]. Financial institutions rely on volatility forecasts to assess portfolio risk and develop trading strategies by hedging against the downside.

One of the earliest efficient models for volatility forecasting is the Autoregressive Conditional Heteroskedasticity (ARCH) model developed by Engle in the 1980s. Engle captured the volatility's time-varying behavior, or the conditional variance, using the past lagged squared returns [1]. The extension of the ARCH model, called the GARCH model, considered both the lagged squared returns and recent historical volatility data to make more accurate predictions [3]. In recent years, advancements in computing techniques and the availability of high-frequency data have resulted in the development of increasingly complex volatility forecasting models with the ability to capture complicated non-linear relationships between the variables. The machine learning algorithms, including Random Forests, Gradient Boosting, and deep learning architectures like Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and Gated Recurrent Unit (GRU) networks have gained popularity in volatility and price forecasting [4].

Despite the advancement of forecasting models, challenges remain in the form of non-linear relationships and unpredictable market events. Researchers continue to explore new methodologies and techniques to enhance the accuracy and robustness of volatility forecasts, with the goal of improving risk management practices and promoting financial stability in global markets.

1.2. OVERVIEW OF FORECASTING MODELS

The traditional time series forecasting models studied in this thesis include the ARCH and the GARCH models. The ARCH model is adept at capturing time-varying volatility in financial time series data by modeling the variance as a function of past squared returns. Building on the ARCH model, GARCH models offer enhanced flexibility by incorporating both lagged squared residuals and lagged conditional variances to capture the volatility more effectively.

On the other hand, machine learning techniques studied here are, Lasso Regression, Random Forest, and Gradient Boosting, which have emerged as powerful tools for forecasting. Lasso Regression employs L1 regularization to encourage only the significant regression coefficients to remain in the model, aiding in feature selection and improving model interpretability [5]. Random Forest, introduced by Breiman (2001), is an ensemble learning technique that improves performance by aggregating multiple predictions made from several decision trees [6]. Gradient Boosting was proposed by Friedman in 2001, and it improves model performance by iteratively building the decision trees while minimizing a loss function [7].

In the domain of deep learning, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (BiLSTM) networks have revolutionized sequence modeling. LSTM networks, proposed by Hochreiter and Schmidhuber (1997), excel at capturing long-range dependencies in sequential data through specialized memory cells instead of the traditional neurons in neural networks [8]. GRU is a simplified variant of LSTM that offer comparable performance with reduced computational complexity (Chung et al., 2014). BiLSTM networks is an extension of LSTM that process input sequences in two directions, enabling the model to capture dependencies from both past and future contexts [9].

1.3. RESEARCH OBJECTIVES

The primary objective of this thesis is to conduct a comprehensive comparative analysis of volatility and price forecasts using various models, including ARCH, GARCH, Lasso Regression, Random Forest, Gradient Boosting, LSTM, GRU, and BiLSTM. The aim is to evaluate the performance and effectiveness of these models in forecasting both volatility and price movements of Bitcoin using its daily closing price data. The existing methodologies in price and volatility forecasting is ambiguous and difficult to reproduce. This research also aims to develop an interactive dashboard that enables users to upload their dataset, train forecasting models using the selected methodology or their own hybrid models built by combining the available methodologies, and analyze the performance metrics and prediction charts to compare different models.

By developing an interactive dashboard, this research aims to make the advanced forecasting techniques more accessible to practitioners and decision-makers in finance. The user-friendly interface of the dashboard allows users to upload their datasets, select the desired forecasting model, and analyze the results without requiring expertise in programming or data science. The interactive dashboard facilitates empirical analysis by enabling users to train forecasting models on their own datasets and assess their performance by analyzing the performance metrics and visualizing the prediction charts.

1.4. OUTLINE

The document is structured into several sections, each focusing on different theories and methodologies related to data analysis and forecasting. Section 1 provides the background study in volatility and price forecasting. It also introduces various forecasting techniques that have been developed over the decades. The research objective for this thesis is also illustrated in this section. Section 2 provides the description of previous research in the field of financial forecasting.

In Section 3, the theory background concerning the various forecating models are described in depth. The first subsection provides the fundamental concepts such as probability density function (pdf), cumulative distribution function (cdf), and expected values. The second part delves into time series analysis, introducing various statistical models for time series data, including white noise, random walk, and autoregressive models. The third part explains the concept of machine learning, starting with linear regression and progressing to more complex ensemble methods such as bagging, random forest, and gradient boosting. Building on that, the forth part explores Neural networks, beginning with an explanation of neurons and activation functions. This subsection then covers forward propagation, loss functions, and the mathematics of backpropagation for updating the network parameters. The next subsection focuses on Sequence Modeling with Recurrent Neural Networks (RNNs), discussing the challenges of traditional RNNs and introducing solutions like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The fifth part concludes with the definition of Bidirectional LSTM (BiLSTM).

Section 4 presents the results, starting with descriptive statistics for Bitcoin, Ethereum, and their volatilities. The section then discusses forecasting methodologies for both price and volatility of Bitcoin, including performance metrics such as RMSE, MAPE, and adjusted R-squared. Results of price forecasting for various models are provided, along with volatility forecasting using GARCH and LSTM. This section ends by presenting the Interactive Dashboard, outlining its elements and providing a screenshot of the homepage. Finally, Section 5 provides the conclusion of the thesis, highlighting the key findings, implications, and future works.

2. LITERATURE REVIEW

Financial volatility and price forecasting have been the topic of interest for financial institutions for decades. The capacity to accurately forecast the evolution of financial market dynamics over time has broadened the scope of research interest in diverse econometrics and machine learning techniques. Traditional econometric models, ARCH and GARCH, developed by Engle and Bollersev respectively have been studied to predict the volatility of various market instruments. Due to the revolution in computational capacity in modern computers have led way to powerful deep learning architectures like LSTM and GRU, which can usually outperform the traditional models in most scenarios. The paper by Ze Shen et al, compares the performance of these conventional econometrics models (GARCH and EWMA) and a machine learning model (Recurrent Neural Network) in forecasting Bitcoin return volatility [10]. The results show that the RNN outperforms both GARCH and EWMA in average forecasting performance measured by MAE performance metric. It also shows that RNN is less efficient at capturing extreme events in Bitcoin compared to the tradtional models.

Gustavo Di-Giorgi et al proposed a new methodology for volatility forecasting in time series by combining GARCH models (EGARCH, IGARCH, TGARCH, ALL-GARCH) with recurrent neural networks (LSTM, BiLSTM, GRU). The results show that the proposed approach, especially using the BiLSTM model, outperforms the individual GARCH and recurrent neural network models in terms of various performance metrics like RMSE, MAE, MAPE, R-squared, and Spearman correlation [4].

Hum Nath Bhandari et al used LSTM to predict the next day closing price of S&P 500 index using nine indicators, including macroeconomic data and technical indicators. They developed single and multi-layer LSTM models and concluded that single layer LSTM outperformed in forecasting the price [11]. Many of the research in volatility and price

forecast use closing price of the asset with sliding window approach to make predictions. However, this paper used eight more features in addition to the closing price to make final forecasts.

Monghwan Seo and Geonwoo Kim developed hybrid forecasting models that combine GARCH family models with machine learning approaches like Artificial Neural Networks (ANN) and Higher Order Neural Networks (HONN). They experimented with various hybrid models using different input variables like GARCH model outputs, Google Trends data, and VIX index and found that the hybrid models based on HONN provided the most accurate forecasts for Bitcoin volatility [12].

3. THEORETICAL BACKGROUND

3.1. PROBABILITY AND DISTRIBUTION

3.1.1. Univariate Random Variable. A random variable is a variable that takes on values corresponding to the outcomes of a random event. Each possible value of a random variable is linked to a probability, and the collection of all potential values along with their probabilities is referred to as the probability distribution of the random variable. Random variables can be univariate, bivariate, and multivariate. Univariate random variable deals with only one variable or attribute. We can analyze the probability distribution, expected value, and its variance as follows:

Cumulative distribution function (CDF):

A cumulative distribution of a random variable X is a function that gives the probability that X is less than equal to x. It is given by $F_X(x) = P(X \leq x)$ for all $-\infty < x < \infty$. The random variable X is continuous if its corresponding cumulative distribution $F_X(x)$ is continuous for all values of x.

Probability density function (PDF):

If X is a continuous random variable with CDF $F_X(x)$, then the probability density function is defined as the derivative of $F_X(x)$ with respect to x. The PDF is given by $f(x) = \frac{d}{dx}F_X(x)$ for all x for which the derivative exists.

Expected Value:

The expected value of a random variable is a measure of the center or average of its probability distribution. It represents the mean or long-term average value that one would expect to obtain if the random experiment were repeated a large number of times. For a continuous random variable with probability density function $f(x)$, the expected value is computed through integration:

$$E[X] = \int_{-\infty}^{\infty} x \cdot f(x)\, dx.$$

Properties of expected value

1. Linearity

$$E[aX + bY] = aE[X] + bE[Y].$$

2. Expectation of a Constant:

$$E[c] = c.$$

3. Expectation of a Function:

$$E[g(X)] = \int_{-\infty}^{\infty} g(x) \cdot f(x)\, dx.$$

Variance:

The variance of a random variable X is a measure of the spread or dispersion of its probability distribution. It is denoted by $Var(X)$ or σ^2 and is defined as the average of the squared differences between each value of X and the expected value ($E[X]$).

For a continuous random variable with probability density function $f(x)$, the variance is obtained through integration:

$$Var(X) = E[(X - E[X])^2] = \int_{-\infty}^{\infty} (x - E[X])^2 \cdot f(x)\, dx.$$

The square root of the variance is called the standard deviation (σ), providing a measure of the typical distance between the values of X and its mean. Variance can also be calculated using the formula:

$$Var(X) = E[X^2] - (E[X])^2.$$

In finance and statistics, variance is commonly used as a measure of volatility. Volatility refers to the degree of variation of a trading price series over time. A higher variance indicates a greater level of price variability, and consequently, higher volatility (Sharpe, 1964).

Property of Variance:

1. Non-negativity:

$$\mathrm{Var}(X) \geq 0.$$

2. Linear combination:

$$\mathrm{Var}(a + bY) = b^2\mathrm{Var}(Y).$$

3.1.2. Bivariate Random Variable. A bivariate random variable refers to a category of random variable that encompasses two variables simultaneously. It comprises a pair of random variables typically represented as (X, Y).

Joint Cumulative distribution function:

The joint cumulative distribution function (CDF) of two random variables X and Y is a function that provides the probability that both X and Y are less than or equal to certain values. It is denoted as $F_{X,Y}(x, y)$ and is defined as:

$$F_{X,Y}(x, y) = P(X \leq x, Y \leq y) \text{ for } x, y \in (-\infty, \infty).$$

The joint CDF is an important tool in understanding the joint behavior of two random variables. It also allows us to get the relation between two random variables through covariances and correlations.

Joint Probability density function:

Like the univariate random variable, the probability density function can be obtained as a mixed partial differentiable with respect to x and y, for all x and y where the partial

derivatives exist.

Mathematically, if $F_{X,Y}(x, y)$ is the joint CDF of random variables X and Y, then the joint PDF $f_{X,Y}(x, y)$ is given by:

$$f_{X,Y}(x, y) = \frac{\partial^2 F_{X,Y}(x, y)}{\partial x \partial y}.$$

The joint PDF satisfies the properties:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{X,Y}(x, y)\, dx\, dy = 1.$$

This integral reflects the total probability under the joint distribution.

Expected value:

The expectation (expected value) of a bivariate continuous random variable (X, Y) is given by:

$$E[X, Y] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x \cdot y \cdot f_{X,Y}(x, y)\, dx\, dy.$$

Here, $f_{X,Y}(x, y)$ is the joint probability density function (PDF) of the bivariate random variables X and Y.

Covariance:

The covariance of random variables X and Y measures the degree of variance of the X and Y together. It is evaluated in terms of expectations as:

$$\text{Cov}(X, Y) = E[XY] - E[X] \cdot E[Y].$$

The covariance can be positive, negative, or zero.

Cov(X, Y) > 0 indicates positive linear relationship between the variables.

Cov(X, Y) < 0 indicates negative linear relationship.

Cov(X, Y) = 0 represents no linear relationship between the variables X and Y. There are

useful properties of covariance:

1. Bilinearity:

$$\mathrm{Cov}(aX + bW, cY + dZ)$$

$$= ac \cdot \mathrm{Cov}(X, Y) + ad \cdot \mathrm{Cov}(X, Z) + bc \cdot \mathrm{Cov}(W, Y) + bd \cdot \mathrm{Cov}(W, Z).$$

2. Symmetry:

$$\mathrm{Cov}(X, Y) = \mathrm{Cov}(Y, X).$$

3. Covariance with Itself:

$$\mathrm{Cov}(X, X) = \mathrm{Var}(X).$$

Correlation:

The correlation of bivariate random variables X and Y, denoted by $\rho_{X,Y}$, is defined as the covariance divided by the standard deviation of X and Y.

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y}.$$

Here, $\mathrm{Cov}(X, Y)$ is the covariance, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y. The range of correlation is given by $-1 \leq \rho \leq 1$.

$\rho_{X,Y} = 1$ indicates perfect positive linear correlation

$\rho_{X,Y} = -1$ indicates perfect negative linear correlation

$\rho_{X,Y} = 0$ indicates no correlation between the random variables X and Y.

Correlation is very important in the prediction of volatility, especially when examining the connection between features that is used to predict the volatility of an asset.

3.1.3. Multivariate Random Variable. The bivariate distribution can be extended to represent more than two random variables. The multivariate cumulative distribution function for a set of n random variables $X_1, X_2, \ldots, X_n$, defined as:

$$F(\mathbf{x}) = P(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n).$$

Here, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ represents a vector of specific values for each random variable. For continuous random variables, the probability density function $f(\mathbf{x})$ is defined as:

$$F(\mathbf{x}) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \ldots \int_{-\infty}^{x_n} f(u_1, u_2, \ldots, u_n)\, du_1\, du_2 \ldots\, du_n.$$

Here, $f(u_1, u_2, \ldots, u_n)$ is the multivariate probability density function.
Multivariate distribution is used to capture the underlying patterns and uncertainties in the sequential data to allow for forecasting.

Gaussian distribution:
Gaussian distribution is a continuous probability distribution that is symmetric around its mean. It is also called the normal distribution and is represented by two parameters: mean and variance. In time series analysis, the normal distribution plays an important role in modeling the distribution of random variables and the associated noise. It simplifies the parameter estimation when using likelihood function, and also allows to model the distribution of forecast error. [13]
The probability density function (PDF) of the Gaussian distribution is given by:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Here, μ is the mean, also known as the location parameter and σ^2 is the variance or the scale parameter of the distribution

3.2. TIME SERIES

3.2.1. Introduction. Time series analysis is a systematic approach addressing the correlations between observations in time. Its impact spans various fields such as economics, where stock market fluctuations and unemployment rates are tracked [13]. This thesis uses time series to forecast the volatility of a cryptocurrency. Cryptocurrencies are one of the most volatile and risky assets to invest in today's market. Forecasting the volatility of these cryptocurrencies using time series and deep learning would enable sound decision making when buying and selling them.

Before doing the analysis, it is essential to first identify whether time series representation is appropriate for a given dataset. This example showcases the annual returns of the S&P 500 spanning from 1920 to 2023 (Figure 3.1) [13]. It illustrates a consistent upward trend in returns over time and allows us to calculate an average return of approximately 7% across this entire period. Additionally, the returns exhibit considerable volatility. Analysis of cryptocurrency is similar to that for a stock market index. Forecasting the volatility of future returns poses a challenge when analyzing this kind of financial data. Various models like ARCH, GARCH, and stochastic volatility models have been created to address these challenges [13]. In this thesis, our objective is to estimate return volatility in similar datasets by employing LSTM alongside GARCH models.

3.2.2. Time Series Statistical Models. Time series analysis aims to create mathematical models that accurately describe sample data, especially data that appears to fluctuate randomly over time. It defines a time series as a set of random variables ordered by their occurrence in time.

Mean function:

The mean function μ_t for a stochastic process $\{X_t\colon \mathrm{t} = 0, 1, 2, \ldots,\}$ is defined as:

$$\mu_t = E(X_t).$$

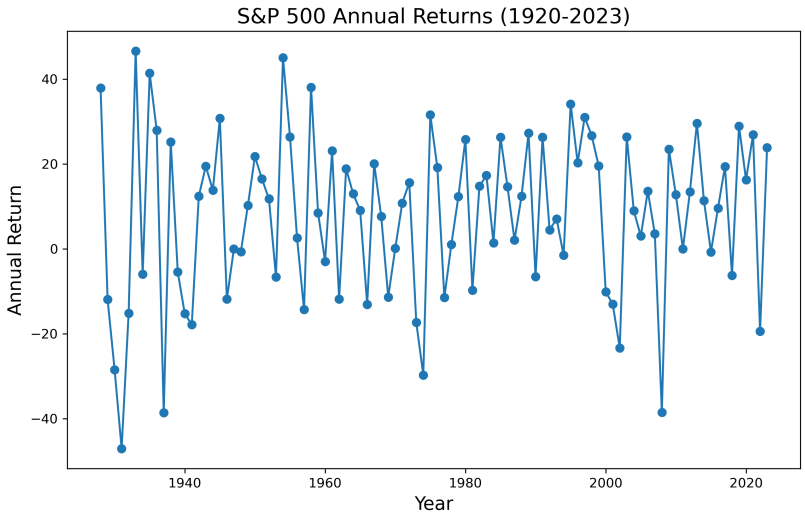

Figure 3.1. S&P 500 Annual Returns from 1920 to 2023

Autocovariance:

Autocovariance measures the linear dependence between observations at different times. The autocovariance ($\gamma_{t,s}$) of an observation at time t and s is given by

$$\gamma_{t,s} = Cov(Y_t, Y_s) = E(Y_t Y_s) - E(Y_t)E(Y_s).$$

Autocorrelation:

The correlation between an observation at time t and s is called autocorrelation ($\rho_{t,s}$) and is defined as:

$$\rho_{t,s} = \frac{Cov(X_t, X_s)}{\sqrt{Var(X_t)\ Var(X_s)}} = \frac{\gamma_{t,s}}{\sigma_t\ \sigma_s}.$$

$\rho_{t,s} = \pm\ 1$ indicates strong linear dependence between X_t and X_s.

$\rho_{X,Y} = 0$ indicates no correlation between X_t and X_s.

Measuring trends:

Time series is a study of patterns and trends over time, which could be stochastic or deterministic. Deterministic trends can be measured with higher certainty using regression

as it is caused by a non random process [13]. An example of a model that measures the linear and quadratic trends are given by:

$$X_t = \beta_0 + \beta_1 t + \varepsilon_t \text{ (Linear trend).}$$

$$X_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \varepsilon_t \text{ (Quadratic trend).}$$

Here, ε_t is the error (innovation) term in the model.

Stochastic trends are caused by random variation and can be represented by a set of random variables $\{X_t : t = 0, 1, 2, \dots\}$. These trends cannot be measured in a deterministic way as presented earlier, but can be analyzed by exploiting the autocorrelation between the observations [13].

3.2.2.1. White noise. White noise is a series of uncorrelated random variables with mean 0 and a finite variance that shows no discernible patterns. A useful white noise series is the Gaussian white noise denoted by $w_t \sim$ i.i.d $N(0, \sigma_w^2)$. This series has a mean 0 and variance σ_w^2, and is often added to the time series model to represent the uncertainty in predictions. White noise is used to model new information as time progresses that cannot be learned from the past, and is fundamental to more complex time series models [13].

Stationarity:
A time series X_t is stationary if it satisfies the following conditions:

- The mean function $E[X_t]$ is constant over time.
- The covariance between X_t and X_s depends only on time lag, which is the difference between the time points t and s, represented by $lag|t - s|$.

A white noise process is stationary because its mean is constant, usually 0, and possesses constant variance given by

$$Var(X_t) = Cov(X_t, X_t) = Cov(X_s, X_s) = Var(X_s).$$

Stationarity is important in the analysis of time series because it makes the statistical analysis more tractable as the probabilistic behavior does not change over time. Prior to building time series models, stationarity is usually induced in non-stationary observations by various methods such as log transformation, differencing, smoothing (Moving averages or exponential smoothing), removing outliers, and so on [13].

3.2.2.2. Random walk. Random walk is a stochastic process where the current observation is obtained by adding current white noise to the previous observation. It is a type of non-stationary time series where the observations are independent of each other, and no discernible patterns are present [13]. It is mathematically described as:

$$X_t = X_{t-1} + \varepsilon_t.$$

Using repeated substitutions, it can also be represented as:

$$X_t = X_0 + a \sum_{i=1}^{t} \varepsilon_i.$$

Mean and Variance of random walk process:

$$E[X_t] = X_0 + \sum_{i=1}^{t} E[\varepsilon_i] = X_0 + t\mu,$$

$$Var(X_t) = \sum_{i=1}^{t} Var(\varepsilon_i) = t\sigma.$$

Here, μ and σ are the constant mean and constant variance of the white noise process.

$$Cov(X_t, X_s) = Cov(\sum_{i=1}^{t} \varepsilon_i, \sum_{i=1}^{s} \varepsilon_i) = Cov(\sum_{i=1}^{t} \varepsilon_i, \sum_{i=1}^{t} \varepsilon_i + \sum_{i=t+1}^{s} \varepsilon_i),$$

$$= Cov(\sum_{i=1}^{t} \varepsilon_i, \sum_{i=1}^{t} \varepsilon_i) = t\sigma.$$

Here, $t < s$ and the covariance only depends on time s.

3.2.2.3. Autoregressive model. Autoregressive (AR) models are obtained by regressing on the past observations to obtain current observation. A p^{th}-order Autoregressive model, AR(p) is defined as follows [13]:

$$X_t = \phi_0 + \phi_1 \, X_{t-1} + \phi_2 \, X_{t-2} + \ldots + \phi_p \, X_{t-p} + \varepsilon_t.$$

Here, ϕ is the model parameter. Thus, an AR(p) model evaluates the current value by regressing on past p values and adding a current white noise process.

First order Autoregressive, AR(1): The first order AR process is mathematically defined as;

$$X_t = \phi_0 + \phi_1 \, X_{t-1} + \varepsilon_t.$$

The stationarity of this model can be preserved by keeping the model parameter $|\phi| < 1$. Setting the model parameter in such a way ensures that the series does not explode as time increases, and the stability is maintained [13].

Model Properties of AR(1) [13]:

Mean function:

$$E[X_t] = \phi_0 + \phi_1 \, E[X_{t-1}].$$

Assuming stationarity, the expectation at time t is equal to the expectation at time $t-1$. So, the expected value will be:

$$E[X_t] = \frac{\phi_0}{1 - \phi_1}.$$

Variance function:

$$Var(X_t) = \phi_1^2 \, Var(X_{t-1}) + Var(\varepsilon_t).$$

The variance is constant over time because of stationarity. So, the variance of the series is given as follows:

$$Var(X_t) = \frac{\sigma_\varepsilon^2}{1 - \phi_1^2}.$$

Autocorrelation function:

The covariance of X_t and X_{t-k} is obtained as,

$$Cov(X_t, X_{t-k}) = \phi_1 \, Cov(X_{t-1}, X_{t-k}).$$

Inductively, the covariance will be:

$$Cov(X_t, X_{t-k}) = \phi_1^k \, Cov(X_{t-k}, Covt - k) = \phi_1^k \, Var(X_t).$$

Now, the lag-k autocorrelation can be defined as:

$$\rho_k = \frac{Cov(X_t, X_{t-k})}{Var(X_t)} = \phi_1^k.$$

Since, $|\rho_k| < 1$, the magnitude of autocorrelation decreases exponentially with lag k. Figure 3.2 shows the exponential decay of autocorrelation with lag.

When $\phi > 0$, the autocorrelations are all positive and decreasing with increasing lag as shown in the following figure.

When $\phi < 0$, the autocorrelations alternate between positive and negative, but still exponentially decreasing with increasing lag.

Heteroskedastic time series models:

The earlier models discussed had stationary random variables, which means constant mean and variance over time. Heteroskedastic refers to a sequence of random variables for which the variance changes as time progresses. In financial modeling, and specifically volatility forecasting, the conditional variance of the series is evaluated using the time series models like Autoregressive conditional heteroskedasticity (ARCH) models to forecast the volatility

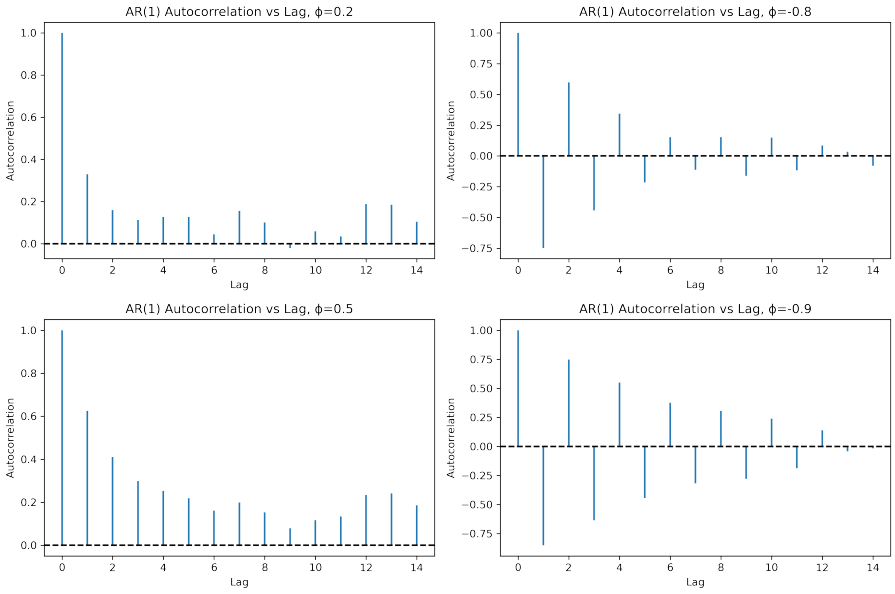

Figure 3.2. Autocorrelation function (ACF) plot for AR(1) model with different model parameters.

over a time period.

For a time series $\{X_t\}$, the conditional variance of X_t given the past values $\{X_{t-1}, X_{t-2}, \dots\}$ measures the uncertainty in deviation of X_t from the conditional mean of the series $E[X_t|X_{t-1}, X_{t-2}, \dots]$.

Returns of the time series:

The returns of a heteroskedastic time series $\{X_t\}$ is defined as [4],

$$r_t = log(X_t) - log(X_{t-1}).$$

The returns r_t is usually a series of uncorrelated random variables with zero mean. The conditional variance of the return is denoted by $\sigma^2_{t|t-1}$, where the variance at time t is conditioned upon the return in time $t-1$. The squared of the returns signifies the amount of volatility, where a series of large squared return may predict a higher volatile period and a quiet period when the series of squared return is relatively small [4].

3.2.2.4. Autoregressive conditional heteroskedastic (ARCH) model. A general ARCH(p) model was introduced by Engle in 1982 and is defined as [1]:

$$\sigma^2_{t|\Omega(t-1)} = \omega + \alpha_1 \, r^2_{t-1} + \alpha_2 \, r^2_{t-2} + ... + \alpha_p \, r^2_{t-p}.$$

Here, $\Omega(t-1)$ contains the information of the series from time $t-p$ to time $t-1$. $\omega > 0$ and $\alpha_1, \alpha_2, \ldots, \alpha_p$ are the non-negative parameters of the model.

ARCH(1) model:

ARCH(1) is the first order model that evaluates the one step conditional variance based on immediate past observation. It assumes that the series is generated as follows:

$$r_t = \varepsilon_t \, \sigma_{t|t-1}.$$

Here, $\{\varepsilon_t\}$ is a sequence of independent and identically distributed random variables with zero mean and unit variance. The ARCH(1) model is defined as:

$$\sigma^2_{t|t-1} = \omega + \alpha \, r^2_{t-1}.$$

Assuming that the return series is stationary, we can take the expectations of this equation to yield:

$$E[\sigma^2_{t|t-1}] = E[\omega] + E[\alpha \, r^2_{t-1}].$$

$$\sigma^2 = \omega + \alpha \, \sigma^2.$$

$$\sigma^2 = \frac{\omega}{1-\alpha}.$$

So, it is necessary and sufficient to have $0 \leq \alpha < 1$ to preserve stationarity [13].

This model can be used to predict k step ahead conditional variance. First, the one step ahead forecast can be obtained by [13]:

$$\sigma^2_{t+1|t} = \omega + \alpha\, r_t^2 = (1-\alpha)\sigma^2 + \alpha\, r_t^2.$$

It can be seen that one step ahead conditional variance is the weighted average of long-run variance and current squared return. Then, the k-step ahead forecast is obtained as [13]:

$$\sigma^2_{t+h|t} = E[r^2_{t+h}|r_t, r_{t-1}, \dots].$$

Iterating the expectation through time, the conditional variance takes the recursive form [13]:

$$\sigma^2_{t+h|t} = \omega + \alpha\, \sigma^2_{t+h-1|t}.$$

3.2.2.5. Generalized autoregressive conditional heteroskedastic (GARCH) model. In the ARCH(p) model, the forecast of conditional variance at time t is represented as the weighted sum of past squared returns up to time $t - p$. The GARCH model, presented by Bollerslev in 1986, also introduces q lags of the conditional variance in addition to the p squared returns. GARCH(p, q) is defined as follows [3]:

$$\sigma^2_{t|\Omega(t-1)} = \omega + \alpha_1\, r^2_{t-1} + \dots + \alpha_p\, r^2_{t-p} + \beta_1\, \sigma^2_{t-1|t-2} + \dots + \beta_q\, \sigma^2_{t-q|t-q-1}.$$

Similar to the ARCH(p) model, the coefficients in a GARCH model must also be nonnegative as the conditional variance must be nonnegative.

A time series of size 500 is simulated from the GARCH(1,1) model with parameters $\omega = 0.1$, $\alpha = 0.2$, and $\beta = 0.7$. The innovation term is obtained from standard normal distribution and the plots are shown in Figure 3.3 and 3.4. Volatility clustering can be observed in the returns plot, where large variations are usually followed by large fluctuations and vice versa.

From the ACF plot, it can be observed lag 1 is highly significant, which means a high correlation with immediate past value.

To describe the necessary condition for a GARCH model to be weakly stationary, the return process is assumed to be weakly stationary. Now, taking expectations on both sides of the model equation of GARCH(1,1) model described earlier:

$$\sigma^2 = \omega + \sigma^2(\alpha_1 + \beta_1).$$

$$\sigma^2 = \frac{\omega}{1 - (\alpha_1 + \beta_1)}.$$

Therefore, $\alpha_1 + \beta_1 < 1$ is necessary for the model to be stationary.

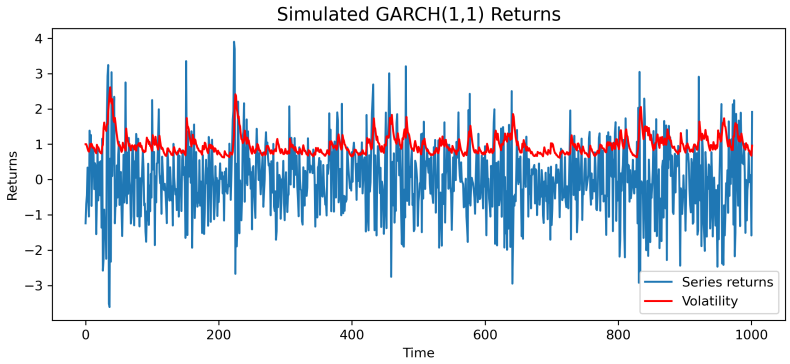

Figure 3.3. Simulated GARCH(1,1) returns with 1000 observations.

If $\alpha_1 + \beta_1 = 1$, then the GARCH(1,1) model is nonstationary and this specific model is called Integrated GARCH(1,1) model [13]. In this case, the conditional variance is evaluated as follows:

$$\sigma^2_{t|t-1} = (1 - \beta)(r^2_{t-1} + \beta\, r^2_{t-2} + \beta^2\, r^2_{t-3} + ...).$$

Here, the conditional variance is an exponentially weighted average of the past squared returns, which means that exponentially less weight is provided to the observation further away in the past [13].

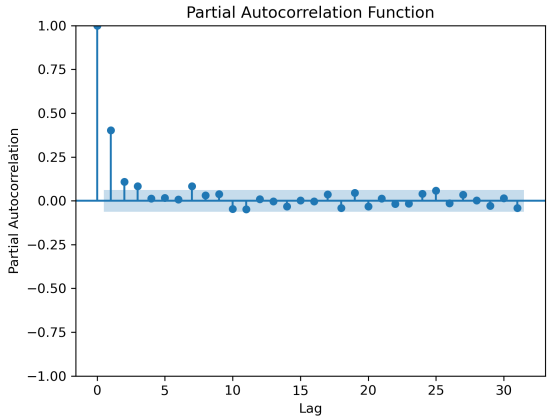

Figure 3.4. Autocorrelation Function of Simulated GARCH(1,1)

3.3. MACHINE LEARNING

3.3.1. Linear Regression. Linear regression is a method used to find the relationship between dependent and independent variables, assuming this relationship is linear. The dependent variable in our case is the closing price at time t, while the independent variables are the past closing prices. The number of independent variables depends on the selected window size, which refers to the number of past observations to consider. After determining the independent variables, we aim to find the best-fitting line (or hyperplane in higher dimensions) that describes the data. This line is determined by coefficients assigned to each independent variable. We adjust these coefficients to minimize the squared difference between the actual data points and the predicted values given by the linear equation.

The linear relationship can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon.$$

Here, y is the dependent variable, $x_1, x_2, \ldots, x_n$ are the independent variables, $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients (parameters) to be estimated, and ε represents the error term.

The coefficients $\beta_0, \beta_1, ..., \beta_n$ are estimated by minimizing the sum of squared residuals below:

$$\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1\, x_{i1} + \beta_2\, x_{i2} + ... + \beta_n\, x_{in}))^2.$$

Here, N is the number of observations.

This can be solved using Ordinary Least Squares method. To minimize this sum of squared residuals, we take the derivative of the expression with respect to each coefficient $(\beta_0, \beta_1, ..., \beta_n)$ and set them equal to zero. The resulting systems of equations are then solved to find the coefficients.

First, the derivative with respect to β_j is given as:

$$\frac{\partial}{\partial \beta_j} \sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1\, x_{i1} + \beta_2\, x_{i2} + ... + \beta_n\, x_{in}))^2 = 0.$$

Evaluating the derivative, we get:

$$\sum_{i=1}^{N} 2[y_i - (\beta_0 + \beta_1\, x_{i1} + \beta_2\, x_{i2} + ... + \beta_n\, x_{in})](-x_{ij}) = 0.$$

There are $n + 1$ sets of equations that can be solved to yield the estimates for the coefficients $\beta_0, \beta_1, ..., \beta_n$.

3.3.2. Lasso Regression. Lasso regression, introduced by Tibshirani in 1996, builds upon linear regression by adding a penalty term to the optimization process. This penalty term helps to prevent overfitting and encourages simpler models by forcing some of the coefficients to be exactly zero. In the estimation of volatility or price, if some of the past closing prices are not significant, the coefficients of those observations would be set to zero. In this way, Lasso performs feature selection by shrinking some coefficients down to zero, effectively removing them from the model [5]. This is particularly useful in sequence modeling where we may consider a large window size, for instance, equal to 50. Some

of the past observations may be highly correlated, in which case their coefficients would be zero and the observations would not contribute to the model. The amount of penalty applied is controlled by a parameter called lambda, which is used to balance between model complexity and goodness of fit [14].

The loss function of Lasso regression can be represented as [14]:

$$\text{Loss} = \text{MSE} + \lambda \sum_{j=1}^{p} |\beta_j|.$$

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in}))^2.$$

Here, λ is the regularization parameter that controls the strength of the penalty term, p is the number of predictors (independent variables), $|\beta_j|$ is the absolute value of the coefficients.

The first part of this objective function is the same ordinary least squares cost function from simple linear regression, and the second part is the penalty term of lasso regression. The minimization of the objective function is achieved through gradient descent.

Gradient descent in optimization of Lasso objective function:

The first step in gradient descent is to find the partial derivatives of the loss function with respect to each coefficient β_j. The gradient of MSE with respect to coefficient β_j is evaluated as [3],

$$\frac{\partial \text{MSE}}{\partial \beta_j} = \frac{1}{n} \sum_{i=1}^{N} \frac{\partial}{\partial \beta_j} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in}))^2.$$

$$= -\frac{2}{n} \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_n x_{in})) x_{ij} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_{ij}.$$

Here, x_{ij} is the value of feature j for data point i.

The gradient of the regularization term is evaluated as,

$$\frac{\partial}{\partial \beta_j} \lambda \sum_{j=1}^{p} |\beta_j| = \lambda \frac{\partial(|\beta_j|)}{\partial \beta_j} = \lambda \text{ sign}(\beta_j).$$

After evaluating the gradients, the values of the coefficients are iteratively updated:

$$\beta_j := \beta_j - \alpha \left(\frac{\partial \text{MSE}}{\partial \beta_j} + \lambda \text{ sign}(\beta_j) \right).$$

Here, α is the tuning parameter typically determined through cross-validation, aiming to minimize the error by selecting the optimal value.

3.3.3. Decision Trees. Decision trees are used for regression and classification tasks. In this work, we use decision trees for regression to forecast the price and volatility of an asset. The decision tree involves segmenting the predictor space into several simpler regions. The prediction for a given observation is given as the mean of the training observations to which it belongs. The process of building the regression tree is as follows [14]:

1. The predictor space is divided into K distinct and non-overlapping regions ($R_1, R_2, ..., R_K$). The regions could have any shape, however, for simplicity, high-dimensional boxes represent them.

2. The prediction of an observation belonging to a particular region R_j is equal to the mean of the observations in that region.

The objective function of a decision tree is given as follows [14]:

$$\text{Minimize} \sum_{j=1}^{K} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2.$$

Here, $\hat{y}_{R_j}$ is the mean of training observations in j^{th} box.

Recursive binary splitting is used to divide the predictor space as follows:

- Recursive binary splitting is an approach that is used to divide the predictor space into distinct and non-overlapping regions [14].
- At each node starting with the root, the algorithm evaluates all possible splits on all features and selects the one that minimizes the sum of squared error.
- The process of splitting is repeated recursively until a stopping criterion is met. The stopping criteria could be the maximum tree depth or a relationship between the number of nodes and samples.

When performing recursive binary splitting at each step, the predictor X_j and a cutoff point s are selected such that splitting the predictor space into regions R_1 and R_2 defined below leads to the lowest sum squared error (SSE) [14].

$$R_1(j, s) = \{X | X_j < s\} \text{ and } R_2(j, s) = \{X | X_j \geq s\}.$$

$$\text{SSE} = \sum_{i:x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2} (y_i - \hat{y}_{R_2})^2.$$

Here, $\hat{y}_{R_1}$ and $\hat{y}_{R_2}$ are the mean responses for training observations in $R_1(j, s)$ and $R_2(j, s)$ respectively.

Cost complexity pruning:

The recursive binary splitting algorithm produces an efficient tree that does well with the training observations. However, the tree might be too bulky and overfitting. It occurs when the tree becomes too complex and captures the noise in the training data, resulting in poor generalization of the test data. Building a smaller tree with fewer terminal nodes may lead to lower variance at the cost of a small amount of bias, leading to a better test performance [14].

Cost complexity pruning is a way to reduce the size of the tree using a penalty term similar to the Lasso regression. The objective function with cost complexity pruning is defined as

follows:

$$SSE = \sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2 + \alpha|T|.$$

Here, $|T|$ is the number of terminal nodes, $\hat{y}_{R_j}$ is the mean response in j^{th} terminal node, and α is the tuning parameter.

Tuning parameter (α):

- The tuning parameter controls the tree's size, represented by the number of terminal nodes. For each value of α, we can get a sequence of subtrees T_α for the original tree T_0.
- If $\alpha = 0$, no penalty term is applied and we get the original tree T_0.
- If α increases, the original tree is more heavily penalized, shrinking the tree T_α.
- Cross-validation is used to select the ideal α from a grid of values that leads to the smallest error.

3.3.4. Bagging. Bagging, or bootstrap aggregating, is a procedure that builds upon decision trees to reduce variance in prediction. In this method, we create B bootstrap samples from the training data set, build regression tree models on the training sets, and average the predictions. [14] It is defined as follows:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x).$$

Here, $\hat{f}_b$ is the prediction for the bootstrap sample b.

Bagging reduces the variance of the prediction by averaging the predictions as follows [14]:

$$Var[\hat{f}_{bag}(x)] = \frac{1}{B} Var[\hat{f}_b(x)].$$

3.3.5. Random Forest. Bagging involves creating trees from bootstrapped samples using the full set of predictors at every split. This usually results in the trees being correlated, which impacts the model's ability to predict the unseen data. If one predictor contributes significantly more than the others in prediction, all the trees would consider that predictor in their first split, resulting in similar trees. Random forest is a variant of bagging where only a set of m from a total of p predictors are used at every split when creating regression trees [6]. This leads to an even larger reduction in variance when the predictions are averaged. Typically feature size, $m = \sqrt{p}$. We can also visualize what features are more influential in predicting the response values using feature importance and partial dependence plots [14].

The following plots (Figure 3.5) show the influential features when predicting the Bitcoin price forecast. The input features are the past closing prices, and each feature represents the price at a certain time t in the past.

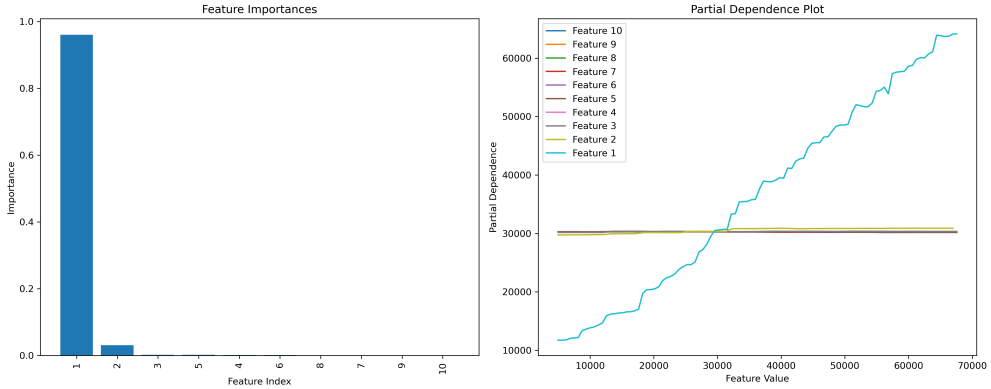

Figure 3.5. Feature importance plot (left) and Partial dependence plot (right) for Random Forest regression model with window size 10.

When predicting the price at time t, the index 1 and index 2 in the first plot represent the prices at times $t-1$ and $t-2$, and so on. Based on this plot, the immediate past observation is highly influential, accounting for around 95% among all the features. Similarly, in the second plot, feature 1 corresponds to the immediate past observation and feature 10 corresponds to the last considered observation for window size 10.

3.3.6. Boosting. Boosting is similar to bagging and random forest as it also involves creating decision trees to make predictions. However, the trees are grown sequentially, where each tree is grown based on the previously grown trees by fitting them on a modified version of the original training data [15]. The modified training data corresponds to the residuals of the previously grown trees. For instance, if a regression decision tree $\hat{f}^1$ is fitted to the original training data, then the residuals of the predictions are given as follows [14]:

$$r_i = y_i - \hat{f}^1(x_i).$$

The residuals are the part of the training data not explained by the first regression tree. Now, a second decision tree $\hat{f}^2$ fits the residuals from the first tree, and the new prediction is represented as the sum of predictions from the first and second trees. A shrinkage parameter is applied to the predictions made from the residuals to control the learning process as follows [14]:

$$\hat{f}(x) = \hat{f}^1(x) + \lambda \hat{f}^2(x).$$

This prediction process from residuals is continued for $b = 1, 2, , , B$. The residual and prediction are updated as follows:

$$\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x).$$

$$r_i = r_i - \lambda \hat{f}^b(x_i).$$

Here, the regression trees at each step b, are fit with d splits, or $d+1$ terminal nodes. Finally, the output from the boosted model can be represented as follows:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

The tuning parameters of the boosting method are as follows [14]:

- Number of trees (B): This parameter determines the amount of time the residuals and the predictions get updated. A large value of B may result in overfitting.
- Shrinkage parameter (λ): This parameter controls the learning rate, with a smaller value resulting in the trees growing slowly.
- Number of splits (d): This parameter determines the complexity of the trees. Usually, we take $d = 1$, which implies that each tree has two leaves.

3.4. NEURAL NETWORKS

3.4.1. Neuron. In a neural network, a neuron takes a set of input values $x_1, x_2, \ldots, x_n$, evaluates their weighted sum, adds a bias, and passes it through an activation function to produce outputs. Neurons are the building block of a neural network, where the interconnected neurons work together to produce the desired results from a set of inputs. A network can have multiple layers and each layer can have multiple neurons, all of which are interconnected through weights. The weight between a connection of neurons represents how an output is related to an input. A bias is also added to the connection to build a more complex relationship between inputs and outputs. In a neural network, the output of a neuron j in layer l is given by [16],

$$z_j^{(l)} = \sigma\left(\sum_{i=1}^{n} w_{ij}^{(l)} x_i^{(l-1)} + b_j^{(l)}\right).$$

Here, $w_{ij}^{(l)}$ is the weight between i-th input and j-th neuron in layer l, $x_i^{(l-1)}$ is the i-th input from the previous layer, $b_j^{(l)}$ is the bias term, and σ is the activation function.

3.4.2. Activation Function. An activation function is an operation applied to the weighted sum of the inputs plus a bias to introduce non-linearity to the output. This allows the neural network to learn complex patterns and relationships in the data and produce meaningful outputs. The four common activation functions are defined as follows [17]:

1. Rectified Linear Unit (ReLU): ReLU takes an input and outputs the maximum value between 0 and the input.

$$\sigma(z) = \max(0, z).$$

2. Sigmoid function: It is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

3. Hyperbolic tangent function (tanh): It produces an output between -1 and 1, and is defined as

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}.$$

4. Softmax function: A softmax function results in an output in terms of probabilities. The output of this function is between 0 and 1. For a vector of outputs $z = (z_1, z_2, \ldots, z_N)$, the softmax function evaluates the output probabilities as

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}.$$

It outputs a value between 0 and 1.

3.4.3. Forward Propagation. Forward propagation involves the calculation of output of each neuron starting from the input layer, propagating through the hidden layers to the output layer [16]. The input layer contains the neurons with input data features. The first hidden layer is evaluated as

$$z_j^{(1)} = \sum_{k=1}^{n} w_{jk}^{(1)} x_k + b_j^{(1)}.$$

$$a_j^{(1)} = \sigma(z_j^{(1)}).$$

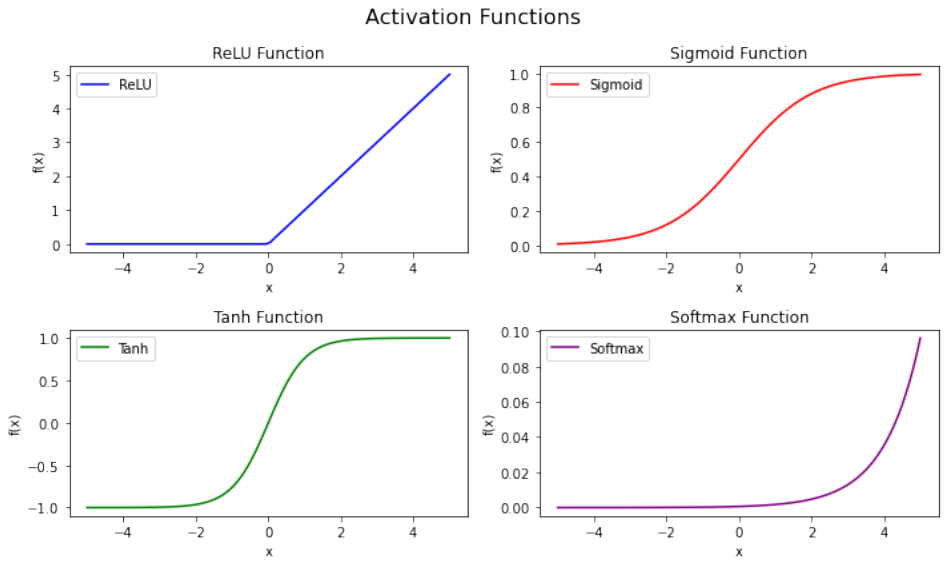

Figure 3.6. Common activation functions used in Neural Network

Here, $z_j^{(1)}$ is the weighted sum for neuron j and $a_j^{(1)}$ is its output after applying the activation function σ.

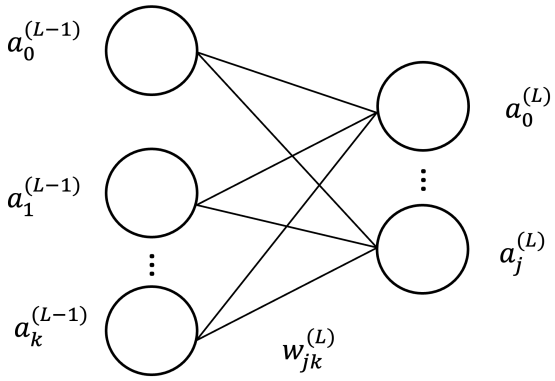

Figure 3.7. A neural network with k and n neurons in layers $L-1$ and L respectively

The output of each neuron in L layers are evaluated as:

$$z_j^{(L)} = \sum_{k=0}^{K} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)}.$$

$$a_j^{(L)} = \sigma(z_j^{(L)}).$$

3.4.4. Loss Function. A loss function measures the difference between true values and predicted values from the neural network. An ideal loss function for volatility forecasting is the sum squared error, which measures the sum of squared difference between the predicted outputs and the actual outputs [18]. The loss function for one training data is defined as

$$C_0 = \sum_{j=0}^{n_L-1} (a_j - \hat{y}_j)^2.$$

If we have N training data, the loss of the network is the average of losses for all training examples.

$$C = \frac{1}{N} \sum_{n=0}^{N} (C_n).$$

Gradient of Loss function:

The gradient gives the direction and the magnitude of steepest increase of the loss function. Minimizing the loss function involves iteratively updating the network parameters in the direction of the negative gradient using a process called gradient descent. The gradient of loss function with respect to the two parameters are the partial derivatives that are evaluated at iteratively until the loss function is minimized. A learning rate is defined when creating the neural network, and is used to update the gradients each iteration [18].

Gradient of Loss with respect to weights ($w_{jk}^{(l)}$) and biases ($b_j^{(l)}$) are given as follows [18]:

$$\text{Weights: } \frac{\partial C}{\partial w_{jk}^{(l)}}.$$

$$\text{Biases: } \frac{\partial C}{\partial b_j^{(l)}}.$$

3.4.5. Backpropagation. When training a neural network, the network parameters that include the weights and biases need to be tuned to minimize the loss function. Backpropagation is an algorithm that accomplishes this task by adjusting the gradients of the loss function with respect to the network parameters (weights and biases). There are four

equations that describe the gradient calculation step of the backpropagation algorithm [18].

Equation 1: Error in output layer

The error of neuron j in layer l, denoted by $\delta_j^{(L)}$, is defined by

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(l)}} = \frac{\partial C}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(l)}} = \frac{\partial C}{\partial a_j^{(L)}} \cdot \sigma'(z_j^{(L)}).$$

Here,

$$\frac{\partial C}{\partial a_j^{(L)}} = 2(a_j^{(L)} - \hat{y}_j).$$

Equation 2: Error of layer l in terms of the error of next layer ($\delta_j^{(l+1)}$)

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_k \frac{\partial C}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_k w_{kj}^{(l+1)}\ \delta_k^{(l+1)}\ \sigma'(z_j^{(l)}).$$

Here,

$$z_j^{(l)} = \sum_k w_{jk}^{(l)}\ a_k^{(l-1)} + b_j^{(l)}.$$

Equation 3: Rate of change of cost with respect to bias

$$\frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)}.$$

$$\Rightarrow \frac{\partial C}{\partial b} = \delta.$$

Equation 4: Rate of change of cost with respect to weight

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{z_j^{(l)}} \cdot \frac{z_j^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)}\ a_k^{(l-1)}.$$

Here,

$$z_j^{(l)} = \sum_k w_{jk}^{(l)}\, a_k^{(l-1)} + b_j^{(l)}.$$

Parameter update:

The weight and bias parameters of the neural network are iteratively updating after each calculation of gradients in the above equations 1 through 4. A hyperparameter called the learning rate, which controls the size of the step taken during gradient descent, is specified when building the neural network. If the learning rate is too small, the minimization of the loss function would converge slowly, however, when the learning rate is too large, the loss function would oscillate without converging to a minimum. The value of learning rate is typically between 0.1 and 0.0001.

The following equations are used when updating the weights and biases after each successful iteration [19]:

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \frac{\partial C}{\partial w_{jk}^{(l)}}.$$

$$b_j^{(l)} = b_j^{(l)} - \alpha \frac{\partial C}{\partial b_j^{(l)}}.$$

3.5. SEQUENCE MODELING WITH RECURRENT NEURAL NETWORK

3.5.1. Recurrent Neural Network. They are a class of neural networks that can analyze sequence data one element at a time while retaining an internal state that stores the past information observed up to that point [17]. Time series forecasting at time t requires building connection to the observed data until $t-1$. RNN does that by maintaining a hidden state h_{t-1} for each time step t. The network parameters are shared across all neurons

as shown in Figure 3.9, and so the RNNs are trained using the same backpropagation algorithm, where the gradients are computed by unfolding the network through time and using the chain rule to evaluate the partial derivatives [17].

RNNs have a cell state, h_t, that is updated at each time step t as a sequence of data, x_t is processed. The cell state is evaluated as a function of input x_t and the old state h_{t-1} [17]. The hidden states are updated as follows:

$$h_t = \sigma(W_{hh}^T\, h_{t-1} + W_{xh}^T\, x_t).$$

Here, W_{hh} and W_{xh} are the weight matrices that need to be optimized when training the network and σ is the activation function.

The output vector of the network is obtained as:

$$\hat{y}_t = W_{hy}^T\, h_t.$$

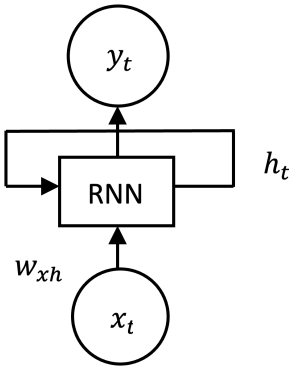

Figure 3.8. A simple RNN structure

Long term dependencies:

Recurrent neural networks use the same weight parameters and operations at each repeated step, giving rise to vanishing or exploding gradient problems. For instance, if a computational graph consists of a path that includes repeatedly multiplying by a weight matrix W,

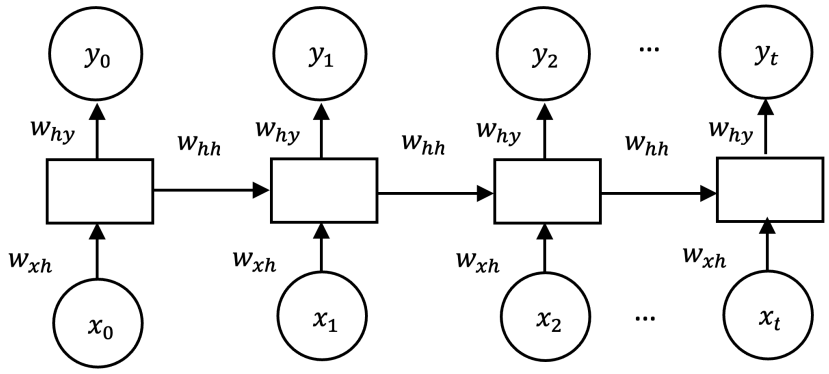

Figure 3.9. Unfolding the graph across time t. The weights are reused for each instance of the graph.

then it is equivalent to multiplying by W^t. The eigendecomposition of W can be represented as $W = V\ diag(\lambda)V^{-1}$, and W^t can be evaluated as [17]:

$$W^t = (V\ diag(\lambda)\ V^{-1})^t = V\ diag(\lambda^t)\ V^{-1}.$$

Two cases follow when any absolute value of eigenvalue is not equal to 1:

- If $\lambda < 1$, the magnitude of λ^t will diminish exponentially with t.
- If $\lambda > 1$, the magnitude of λ^t will explode as the number of repeated operations (t) increases.

The exploding and vanishing gradient problem arises as the gradients are scaled according to $diag(\lambda^t)$. In the case of vanishing gradients, it is difficult to identify which direction the parameters should change to minimize the cost function, while exploding gradients makes the learning process unstable [17].

Assuming that the recurrent neural network is stable and there is no problem of exploding or vanishing gradient, there is still an issue of long term dependency of the recurrent neural network. The weights assigned to past information decrease exponentially as the hidden state is iteratively updated. Because of this reason, the network is unable to keep track of

earlier information which might be useful to give current predictions. An effective way to solve this issue is by using gated recurrent neural networks called long short term memory (LSTM), which utilizes both long term and short term information to give an output [17].

3.5.2. Long Short Term Memory. LSTM has the basic structure similar to RNN, but the neurons are replaced by memory blocks with each memory block containing three non-linear units called gates [8]. A single LSTM unit (Figure 4.2) makes decision by considering the current input, previous output and previous memory and it generates a new output and updates the memory. The three gates are called input gate, forget gate, and output gate.

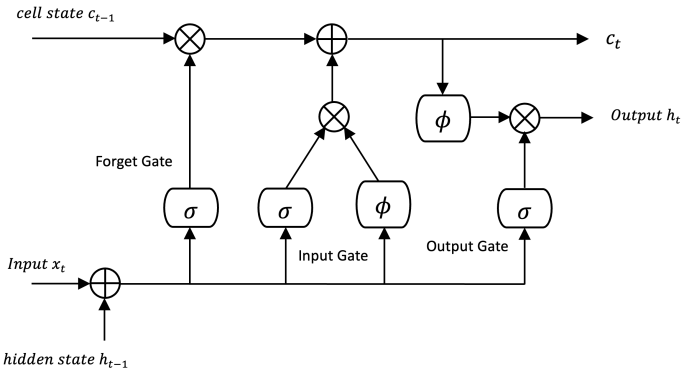

Figure 3.10. A block diagram of LSTM unit

The gate operations are defined as follows [4]:

Forget gate:

A forget gate controls the amount of information kept (or forgotten) from the previous cell state and is defined as:

$$f_t = \sigma(W_f\, x_t + U_f\, h_{t-1} + b_f).$$

Input gate:

The input gate controls the amount of information stored in the cell state from the current input. The input operation is defined as:

$$i_t = \sigma(W_i\, x_t + U_i\, h_{t-1} + b_i).$$

Output gate:

An output gate controls the amount of cell state used to describe the output in terms of the new hidden state h_t.

$$o_t = \sigma(W_o\, x_t + U_o\, h_{t-1} + b_o).$$

Hidden state:

It is the output of a LSTM cell at a time step t and defined as:

$$h_t = \phi(c_t) \cdot o_t.$$

Cell state:

The cell state stores the long-term memory that is passed with each time steps. It can selectively add or remove information through the use of gates and is defined as:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t,$$

where

$$g_t = \phi(W_c\, x_t + U_c\, h_{t-1} + b_c).$$

Here, W_f, W_i, W_o, W_c, U_f, U_i, U_o, U_c are the weight matrices, b_f, b_i, b_o, b_c are the corresponding biases. σ and ϕ are sigmoid and tanh activation functions respectively. The initial values are $c_0 = 0$ and $h_0 = 0$.

A single LSTM unit does not produce the desired result. Several units need to be used together as shown in the following figure. In this study, we used between 5 to 15 LSTM units to train the model.

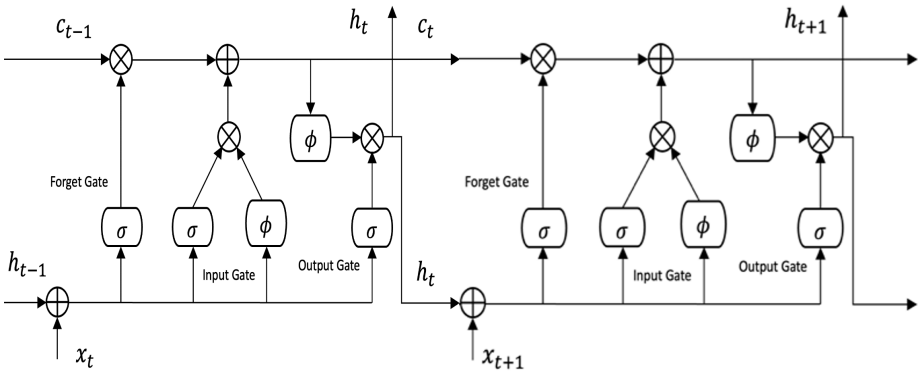

Figure 3.11. Information flow between two LSTM units from time t to $t + 1$

3.5.3. Gated Recurrent Unit. A GRU has simpler architecture compared to LSTM, consisting of two gates: update gate and reset gate [20]. A LSTM unit has a separate cell state that retains long term information to update hidden state, while GRU does not have such a cell state. Instead a GRU unit updates their hidden state directly, containing both short term and long term state in a single unit. Because of a simpler design of GRU, it is computationally less extensive to train compared to LSTM. In this work, both LSTM and GRU are used to train and forecast the sequential data. The gate operations are defined as follows:

Update gate:

It determines the amount of past information to keep and the amount of hidden state to add. The gate operation is as follows [4]:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z).$$

Reset gate:

It determines the amount of past hidden state to forget (or, to consider) and its operation is defined as:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r).$$

Hidden state:

The hidden state of a GRU unit is defined as:

$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t \cdot h_{t-1}) + b_h).$$

Output state:

The output of the GRU unit is given as:

$$h_t = h_{t-1} \cdot (1 - z_t) + z_t \cdot \tilde{h}_t.$$

Here, $W_z, W_r, W_h, U_z, U_r, U_h$ are the weight matrices, b_z, b_r, b_h are the corresponding biases.

3.5.4. Bidirectional Long Short Term Memory. A BiLSTM extends the capabilities of an LSTM network by combining both past and future information. It processes input data in two directions, forward and backward, capturing dependencies in both directions. After processing input sequences, the outputs of the forward and backward LSTM layers at each time step t are combined. These combined outputs are either passed through additional layers for further processing or used directly to make predictions [4].

The parameters of BiLSTM, including the weights and biases of the LSTM units, are trained using backpropagation algorithm described earlier. Figure 3.13 shows the block diagram of BiLSTM with three units.

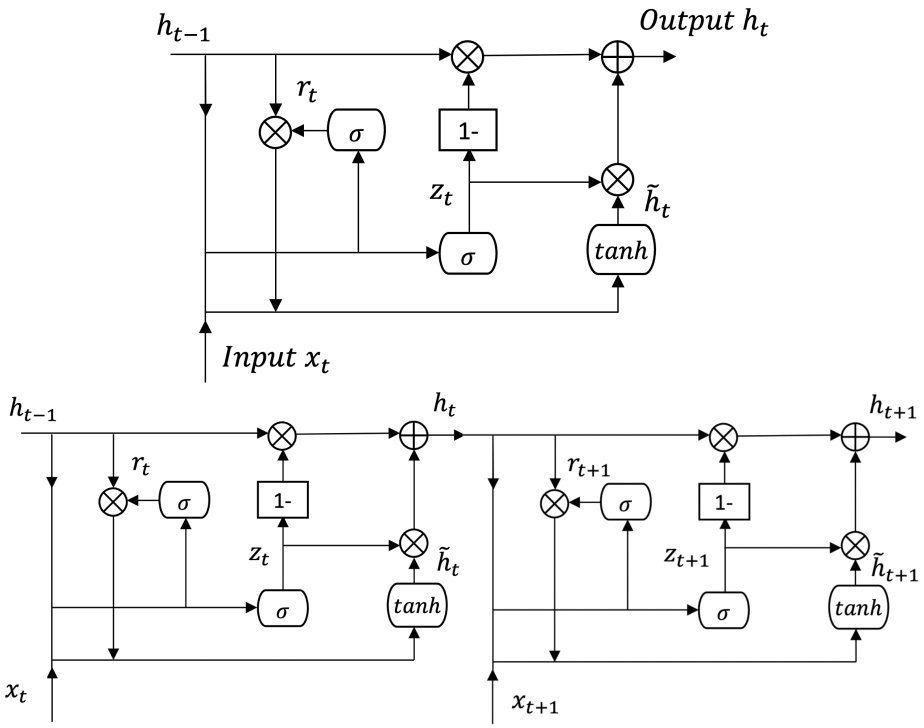

Figure 3.12. A block diagram of GRU unit (top) and information flow between two GRU units from time t to $t + 1$ (bottom)

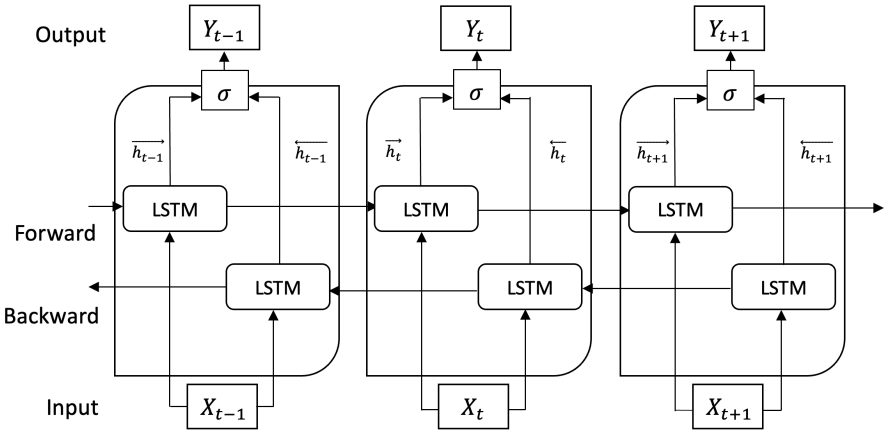

Figure 3.13. BiLSTM architecture with three BiLSTM units for inputs at $t - 1$, t, and $t + 1$

4. RESULTS AND DISCUSSION

4.1. DESCRIPTIVE STATISTICS OF DATA

The descriptive statistics of Bitcoin and Ethereum were analyzed and displayed in Table 4.1. The range of Bitcoin data considered is from September 17, 2014 to March 5, 2024, totaling 3458 trading days. During this range, the minimum price was $178.10 and the maximum was $68330.41. The standard deviation, which is used to measure volatility of an asset, is equal to $16679.57. This is remarkably high compared to other assets today, making it one of the most volatile asset. The skewness of Bitcoin is 1.14, which suggests that the distribution of Bitcoin returns is moderately right-skewed. This indicates that there might be more positive returns compared to negative returns, with the tail of the distribution longer on the positive side. The kurtosis value of 0.22 indicates that the distribution of Bitcoin returns is relatively close to a normal distribution (kurtosis = 0), but with slightly lighter tails and a flatter peak compared to a normal distribution.

The volatility of bitcoin returns at time t was evaluated as the standard deviation of the past seven day returns. Since volatility requires the past seven returns, we only have the volatility values starting from the 8^{th} day. The minimum and maximum volatility are 1.02 and 4476.66 respectively while the standard deviation is 696.65, indicating high fluctuations as expected. It has a skewness value of 2.25, which suggests that the distribution is significantly right-skewed. This indicates that there might be more instances of high volatility compared to low volatility, with the tail of the distribution longer on the high volatility side. The kurtosis of bitcoin volatility is 5.57, which indicates that it has leptokurtic distribution, which means heavier tails and a sharper peak compared to a normal distribution. This suggests extreme volatility compared to the normal distribution.

Table 4.1. Descriptive statistics for Bitcoin, Ethereum, and their corresponding volatilities

	Bitcoin	Bitcoin Vol	Ethereum	Ethereum Vol
Count	3458	3451	2309	2302
Mean	15204.46	491.95	1277.23	50.02
Median	8493.89	206.08	1107.07	27.42
Standard dev	16679.57	696.65	1127.68	59.15
Minimum	178.10	1.02	84.31	0.70
Maximum	68330.41	4476.66	4812.09	504.16
Skewness	1.14	2.25	0.87	2.34
Kurtosis	0.22	5.57	-0.05	8.29

Ethereum is the second dataset that was considered with the range between November 9, 2017 to March 5, 2024, totaling 2309 trading days. During this range, the minimum price was $84.31 and the maximum was $4812.09. The deviation from the mean of $1277.23 is evaluated as $1127.68. Ethereum has a skewness value of 0.87, suggesting a moderately right-skewed distribution. This indicates a similar pattern to Bitcoin, with more positive returns compared to negative returns, and a longer tail on the positive side. It has a kurtosis value of -0.052, which indicates that the distribution is close to a normal distribution, but with slightly lighter tails and a flatter peak.

The volatility of Ethereum returns at time t was also evaluated as the standard deviation of the past seven day returns. The minimum and maximum volatility are 0.70 and 504.16 respectively while the standard deviation is 59.15, indicating high swings similar to Bitcoin. It has a skewness value of 2.34, suggesting significantly right-skewed distribution of Ethereum volatility. This indicates a similar pattern of more instances of high volatility compared to low volatility similar to Bitcoin volatility. It has a kurtosis value of 8.29, which indicates that the distribution is highly leptokurtic, with even heavier tails and a sharper peak compared to Bitcoin volatility. This suggests that extreme volatility events are even more common in Ethereum compared to Bitcoin.

4.2. FORECASTING METHODOLOGY

In this thesis, we forecast price and volatility of Bitcoin using various time series and machine learning models. The price and volatility forecasting methodology are described separately as follows [4]:

4.2.1. Price Forecast.

- The Bitcoin dataset is obtained from Yahoo finance as a downloaded csv file.
- The available features in the dataset are: Open, High, Low, Close, Adj Close, and Volume. Only the closing price (Close) is chosen for our analysis.
- The dataset is checked for any missing values, and split into training (60%), validation (20%), and test (20%) sets.
- We forecast current price using the past closing prices. The number of past observations is defined as window size. The optimal window size is estimated by comparing the performance metrics of the models for the window sizes between 1 and 50.
- The trained models have several model parameters that need to be pre-defined, called the hyper-parameters. These parameters are selected based on the model performances similar to how the optimal window size is obtained.
- After optimizing the window size and hyper-parameters, prediction results are obtained including various charts and performance metrics.
- The performance metrics of various models obtained in the previous step are compared. The models used here are: Lasso Regression, Random Forest Regression, Gradient Boosting, LSTM, GRU, and BiLSTM.

4.2.2. Volatility Forecast.

- The closing price extracted in Price forecast procedure is also used to forecast volatility. The log returns of the closing price are evaluated to introduce stationarity to the dataset. Stationarity is assumed when using ARCH and GARCH models to estimate volatility.
- The volatility of bitcoin returns at time t is evaluated as the standard deviation of the past seven day returns. Volatility is defined as the seven-day moving standard deviation through time.
- Partial Auto Correlation Function (PACF) plot is created to analyze the autocorrelation between observations through time. This is helpful in determining the number of past observations to consider when training the model.
- The models are trained with the optimal window size identified using the PACF plot. The models trained to forecast volatility are: GARCH, LSTM, GRU, and BiLSTM.
- Rolling forecast is generated for the test set. Based on the forecasts, the performance metrics are evaluated and compared between the models defined in the previous step.

4.3. PERFORMANCE METRICS

The performance metrics used to evaluate and validate the trained models are as follows [4]:

1. Root Mean Square Error: It is defined as the square root of the average of squared errors.
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$
2. Mean Absolute Percentage Error: It is defined as the average of absolute percentage difference between the predicted and actual values.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%.$$

3. Coefficient of determination: It is defined as the proportion of variation in the dependent variable (current price) that is explained by the independent variables (past prices).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

Here, $\hat{y}_i$ is predicted value, y_i is true value, $\bar{y}$ is the mean of the true values, and n is the number of observations.

We used adjusted coefficient of determination to account for the number of predictor variables (window size) in the model. The optimal window size may vary with the models trained.

$$\text{Adjusted } \mathrm{R}^2 = 1 - \left(\frac{n-1}{n-p-1}\right) \times (1 - R^2).$$

4.4. TRAINING THE MODEL: PRICE FORECAST

4.4.1. LSTM Model. The LSTM model is trained using the bitcoin dataset to predict the current closing price based on the historical prices. The model configuration of the LSTM model is as follows:

- Number of LSTM units: 14
- Optimizer: Adam (A stochastic gradient-based optimization algorithm)
- Loss metric: mean squared error and r squared
- Activation function: ReLU
- Number of epochs: 25

- Range of dataset: 08/05/2020 to 03/05/2024
- Training size: 60%, Validation size: 20%, and Test size: 20%

The model is trained using different window sizes (1 to 50), and the best model is selected based on the highest adjusted R squared and least mean square error. The Optimal window size estimation using these performance metrics are shown in Figure 4.1 and 4.2.

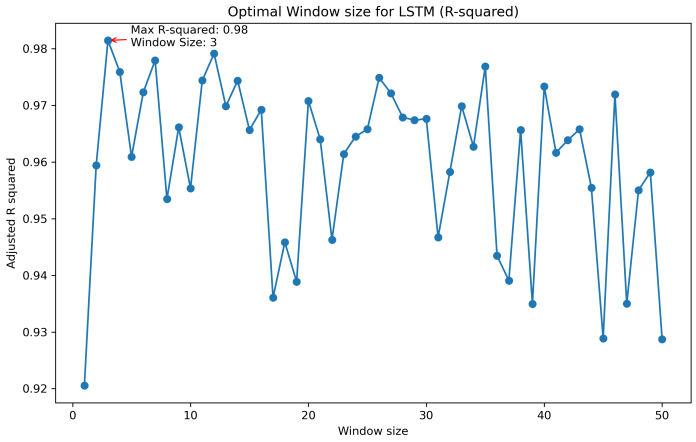

Figure 4.1. Optimal window size for LSTM model based on Adjusted R-squared

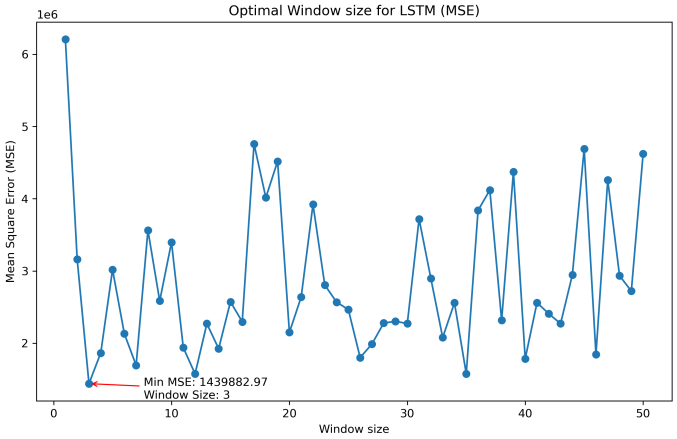

Figure 4.2. Optimal window size for LSTM model based on Mean squared error

From the plots, the optimal window size for LSTM model is equal to 3. This window size equal to 3 is used to train the LSTM model and predictions are evaluated. The prediction plot (Figure 4.3) and performance metrics (Table 4.2) are obtained.

Figure 4.3. LSTM Price Forecast for Window size = 3

Table 4.2. LSTM performance metrics for Window size = 3

	Train	Test
RMSE	1807.64	1156.56
MAE	1333.64	745.52
MAPE	3.98	1.98
R-squared	0.986	0.982

4.4.2. GRU Model. The GRU model is trained similar to the LSTM model and has the same model configuration. The only difference is that the 14 LSTM units are replaced by 14 GRU units. The GRU model is also trained for different window sizes (1 to 50), and the best model is selected based on the highest adjusted R squared and least mean square error. The Optimal window size estimation using these performance metrics are shown in Figure 4.4 and 4.5.

From the plots, the optimal window size for GRU model is equal to 2 (compared to 3 for LSTM). This window size equal to 2 is used to train the GRU model and predictions are evaluated. The prediction plot (Figure) and performance metrics (Table 4.3) are obtained.

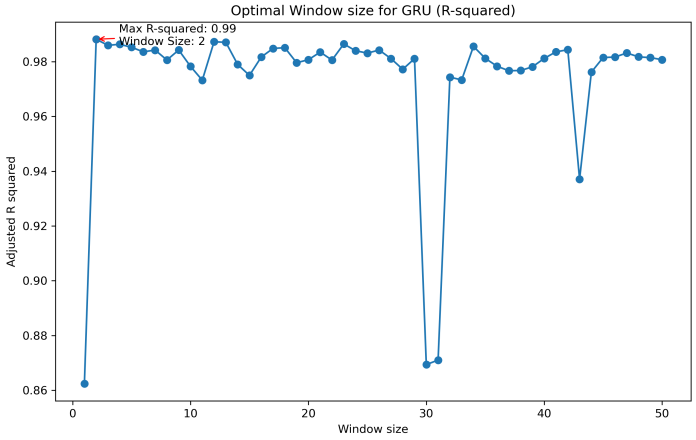

Figure 4.4. Optimal window size for GRU model based on Adjusted R-squared

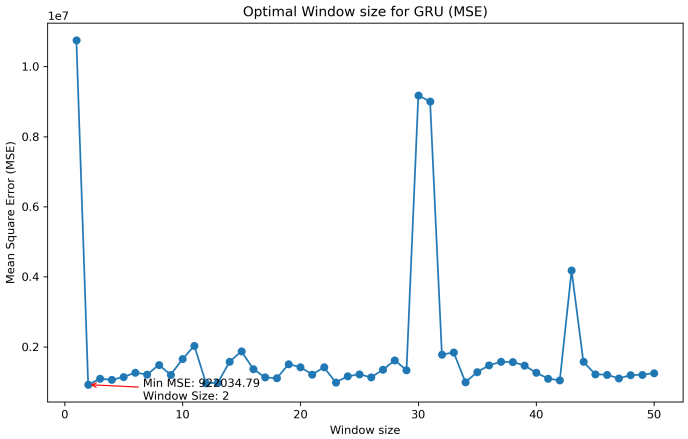

Figure 4.5. Optimal window size for GRU model based on Mean squared error

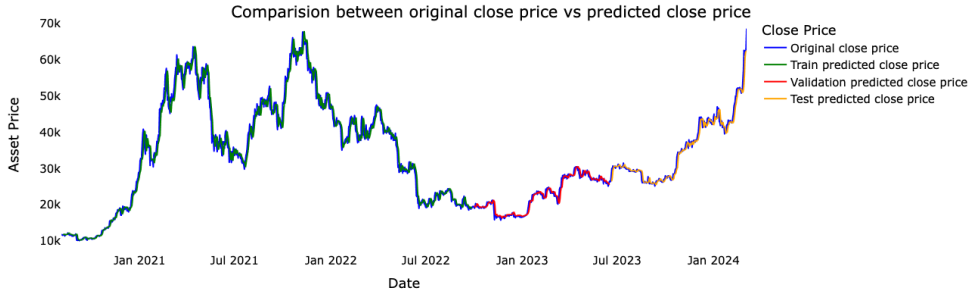

Figure 4.6. GRU Price Forecast for Window size = 2

Table 4.3. GRU performance metrics for Window size = 2

	Train	Test
RMSE	1595.86	986.52
MAE	1106.52	612.11
MAPE	2.99	1.62
R-squared	0.989	0.987

4.4.3. BiLSTM Model. The BiLSTM model is trained similar to the LSTM and GRU models and has the same model configuration. It is also trained for different window sizes (1 to 50), and the best model is selected based on the highest adjusted R squared and least mean square error. The Optimal window size estimation using these performance metrics are shown in Figure 4.7 and 4.8.

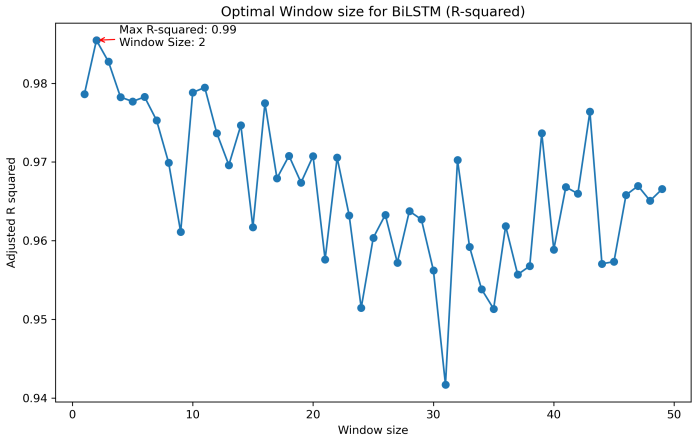

Figure 4.7. Optimal window size for BiLSTM model based on Adjusted R-squared

From the plots, the optimal window size for BiLSTM model is equal to 2, which is similar to the GRU model . This window size equal to 2 is used to train the BiLSTM model and predictions are evaluated. The prediction plot (Figure 4.9) and performance metrics (Table 4.4) are obtained.

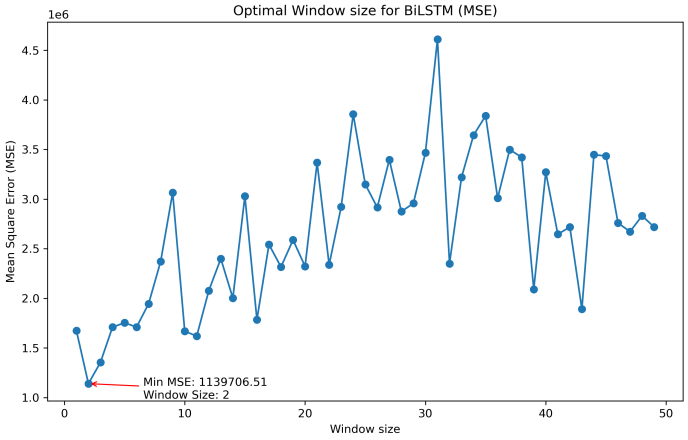

Figure 4.8. Optimal window size for BiLSTM model based on Mean squared error

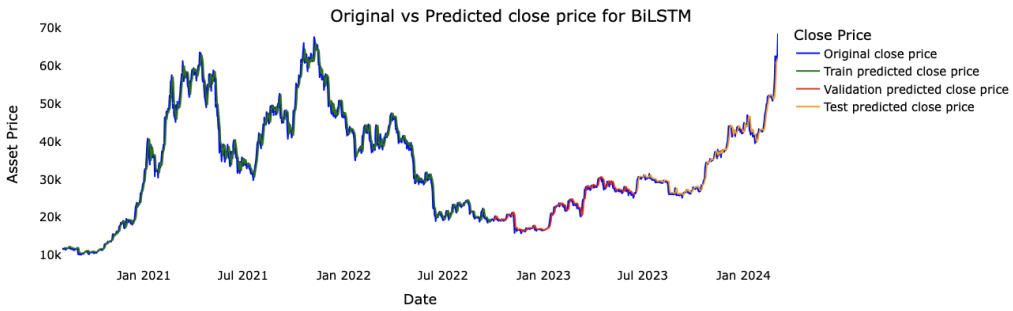

Figure 4.9. BiLSTM Price Forecast for Window size = 2

Table 4.4. BiLSTM performance metrics for Window size = 2

	Train	Test
RMSE	11601.72	968.16
MAE	1142.39	669.26
MAPE	3.36	1.86
R-squared	0.989	0.988

4.4.4. Random Forest Regression. The random forest model is trained using the following model configuration:

- Number of decision trees: 100
- Number of predictors (m): $\sqrt{p}$
- Loss metric: mean squared error
- Range of dataset: 08/05/2020 to 03/05/2024
- Training size: 80% and Test size: 20%

Similar to the previous deep learning models, Random forest is trained using different window sizes (1 to 50), and the best model is selected based on the highest adjusted R squared. The Optimal window size estimation using this performance metrics is shown in Figure 4.10.

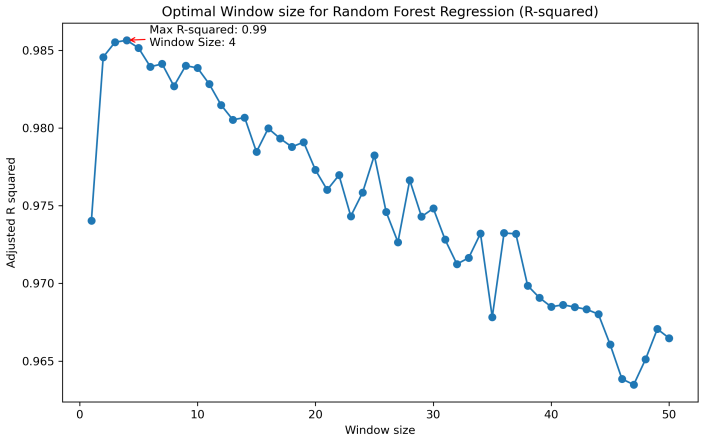

Figure 4.10. Optimal window size for Random Forest Regression model based on Adjusted R-squared

From the plot, the optimal window size for Random forest model is equal to 4. For the window sizes greater than 4, the test accuracy of the model slowly decreases. The window size equal to 3 is used to train the Random forest regression model and predictions are evaluated. The prediction plot (Figure 4.11) and performance metrics (Table 4.5) are obtained.

Figure 4.11. Random Forest Price Forecast for Window size = 3

Table 4.5. Random Forest performance metrics for Window size = 3

	Train	Test
RMSE	502.06	1061.59
MAE	320.22	793.57
MAPE	1.06	2.27
R-squared	0.999	0.985

4.4.5. Gradient Boosting. The gradient boosting model is trained using the following model configuration:

- Number of decision trees: 100
- Shrinkage parameter: 0.3
- Number of splits: 6
- Loss metric: squared error

- Range of dataset: 08/05/2020 to 03/05/2024
- Training size: 80% and Test size: 20%

Gradient boosting is trained using different window sizes (1 to 100), and the best model is selected based on the highest adjusted R squared. The Optimal window size estimation using this performance metrics is shown in Figure 4.12:

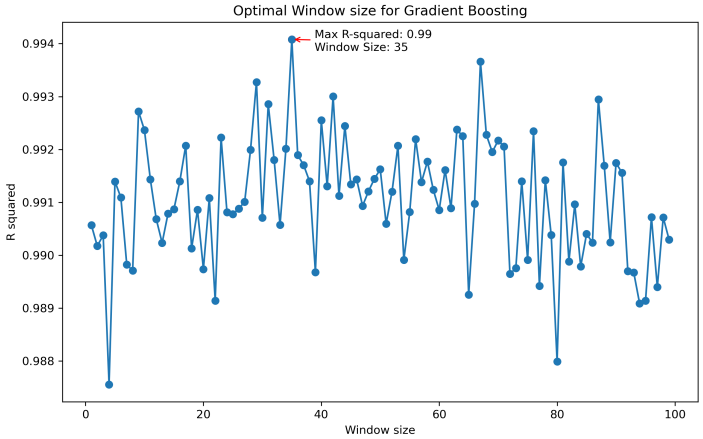

Figure 4.12. Optimal window size for Gradient Boosting model based on Adjusted R-squared

From the plot, the optimal window size for Gradient boosting model is equal to 35, which is significantly higher compared to the previous models. The window size equal to 35 is used to train the Gradient boosting model and predictions are evaluated. The prediction plot (Figure 4.13) and performance metrics (Table 4.6) are obtained.

Table 4.6. Gradient Boosting performance metrics for Window size = 35

	Train	Test
RMSE	60.61	1157.15
MAE	42.26	838.89
MAPE	0.15	2.23
R-squared	1.000	0.982

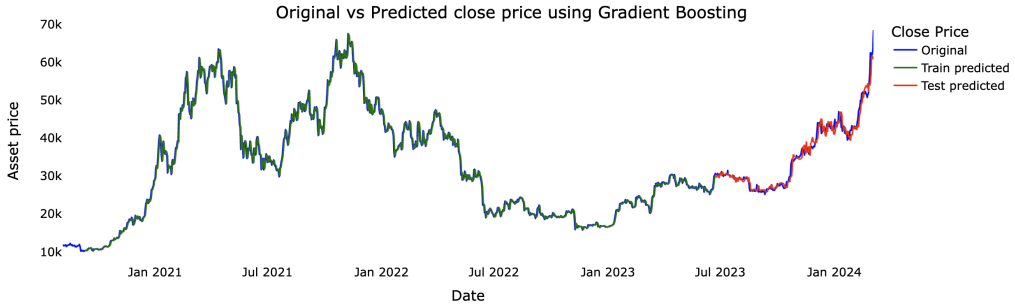

Figure 4.13. Gradient Boosting Price Forecast for Window size = 35

4.4.6. Random Forest and Gradient Boosting Hybrid. In this hybrid model, the Bitcoin price data is predicted through the random forest model, and the residuals are extracted. The residuals are then predicted using the gradient boosting model. The final prediction is made by adding the first and second predictions.

The hybrid model is trained alongside Random forest and Gradient boosting using different window sizes (1 to 100), and the best model is selected based on the highest adjusted R squared and the least Mean absolute percentage error. The Optimal window size estimation using this performance metrics is shown in Figure 4.14 and 4.15.

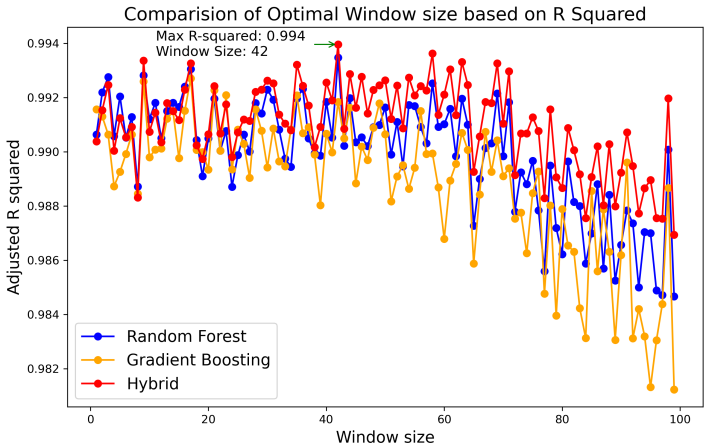

Figure 4.14. Comparison between models based on Adjusted R squared for different window sizes

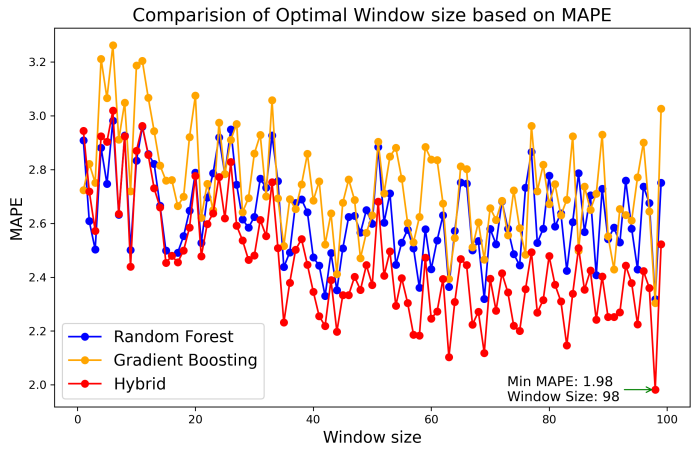

Figure 4.15. Comparison between models based on MAPE for different window sizes

It is evident from the comparison plots that the hybrid model (red) outperforms both Random forest and gradient boosting in terms of both the adjusted R squared and MAPE performance metrics. The best hybrid model with maximum adjusted R squared (0.994) for the test set is achieved for window size equal to 42. However, minimizing the mean absolute percentage error leads to the selection of the hybrid model with window size 98. This is significantly large window size considering that our sample size is 1476. Table 4.7 provides the additional performance metrics for the optimal window size of 42, where it can be observed that the hybrid model outperforms the others in terms of all four metrics.

Table 4.7. Comparison between Random Forest, Gradient Boosting, and their hybrid for the optimal Window size = 42

	Random Forest	Hybrid	Gradient Boosting
RMSE	1163.01	1093.38	1257.04
MAE	792.01	734.47	840.34
MAPE	2.42	2.22	2.52
R-squared	0.993	0.994	0.992

4.5. TRAINING THE MODEL: VOLATILITY FORECAST

4.5.1. Simulated GARCH model. A simulated data was created using the GARCH(1,1) process with 1000 observations. The model parameters used to create the simulated data are: $\omega_1 = 0.1$, $\alpha_1 = 0.2$, $\beta_1 = 0.7$. The GARCH(1,1) model was created to make the forecast and the model summary is given in Table 4.8.

Table 4.8. GARCH (1,1) Model Summary for simulated data

	Coefficient	P-value	95% Conf. Int
ω	0.0349	0.1500	[-0.0126, 0.0825]
α_1	0.1370	0.0041	[0.0434, 0.231]
β_1	0.8217	0.0000	[0.680, 0.963]

The trained model parameters obtained are: $\omega_1 = 0.0349$, $\alpha_1 = 0.1370$, $\beta_1 = 0.8217$. Some similarities and differences between the trained model parameters and the actual model parameters can be observed. The 95% confidence interval of the parameter estimates contains the actual model parameters except for ω. The volatility prediction of the test set using GARCH(1,1) was obtained. Figure 4.16 shows the variation between the actual volatility and the predicted volatility. The difference is because of the way the

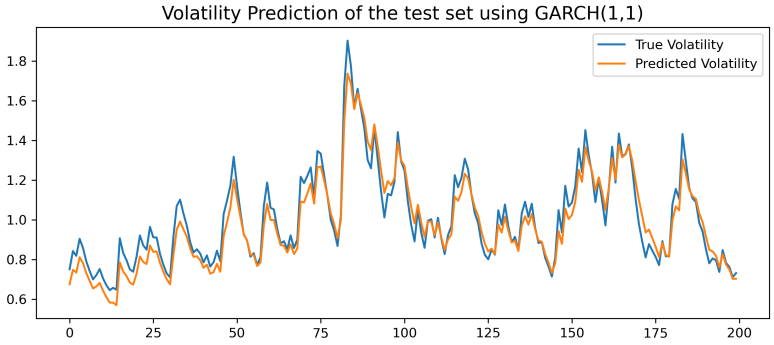

Figure 4.16. Volatility prediction of a simulated data using GARCH(1,1).

volatility is defined in GARCH model. It is defined as the product of an error term and the series. The error term is obtained from standard normal distribution. The coefficient of determination for this model is 0.92.

4.5.2. Simulated GARCH-LSTM Model. The data generated from GARCH(1,1) process was also trained using LSTM model. Two layers, each with 16 LSTM units, were used to train 60% of the observation, while 20% was used for validation and the remaining 20% for test. The model was trained for 100 epochs and predictions were obtained. The predictions were plotted against the actual values and shown in Figure 4.17.

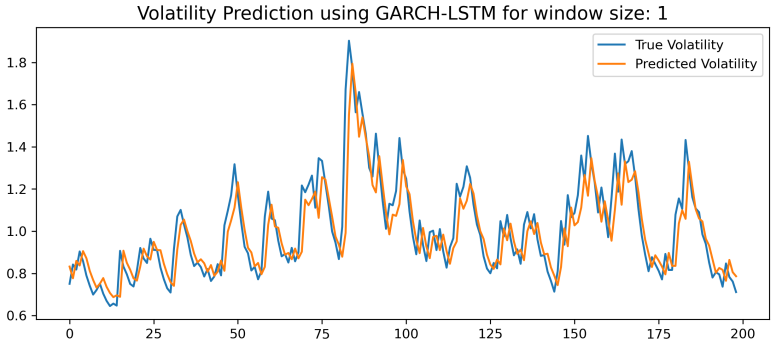

Figure 4.17. Volatility prediction of a simulated data using GARCH-LSTM.

Only the immediate past volatility was used to predict the current value. GARCH (1,1) model described earlier used both the immediate past volatility as well as the past returns to predict the current volatility. This could be the reason why GARCH (R^2 = 0.92) performed better than LSTM (R^2 = 0.72).

4.5.3. GARCH model. The bitcoin dataset was used to create a GARCH model. Only the higher volatility period between February 19, 2021 and March 5, 2024 was chosen to improve predictions. A PACF plot (Figure 4.18) was created to identify the number of past observations to consider in forecast. It can be seen that lag 1 is highly significant. Lag 2 also appear to be significant, however, the performance does not significantly improve when adding the second lag term. So, we continued the analysis based only on the immediate past observation (lag 1), and the model summary is shown in Table 4.9.

Table 4.9. GARCH (1,1) Model Summary for Bitcoin volatility

	Coefficient	P-value	95% Conf. Int
ω	0.0434	0.0164	[0.0079, 0.0788]
α_1	0.4017	0.0000	[0.296, 0.508]
β_1	0.5983	0.0000	[0.472, 0.724]

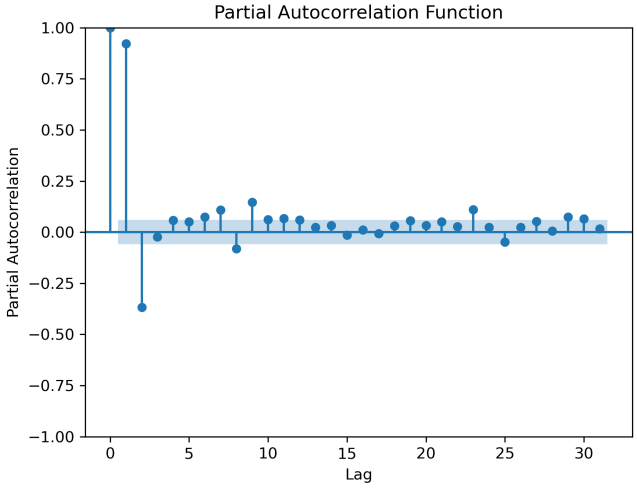

Figure 4.18. PACF plot for bitcoin dataset.

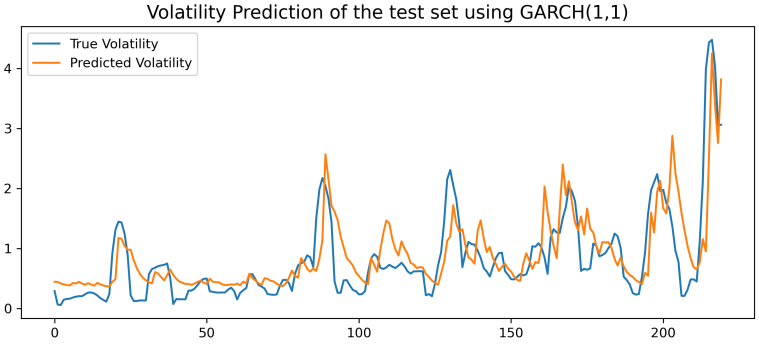

Figure 4.19. Volatility prediction of bitcoin using GARCH model

The rolling forecasts were made by re-training the GARCH model after each time step. The forecasts and actual values are plotted and shown in Figure 4.19. The volatility predictions made by GARCH model are not very accurate with a coefficient of determination equal to 0.52 for the test set. This means that we cannot effectively estimate the bitcoin volatility based on the immediate past observation using a GARCH model. This could be because the series data that is used to make volatility prediction has an underlying error term obtained from a standard normal distribution. However, the noise in volatility may not be effectively defined by the normal distribution. If the noise were normally distributed, we would have been able to obtain better predictions similar to the simulated garch returns, which has an accuracy of 92%.

4.5.4. Deep Learning Models (LSTM, GRU, BiLSTM). The volatility was estimated using an LSTM model with one layer containing 64 units. The model was trained for 200 epochs until the validation and training error had minimized. The LSTM model achieved significantly better performance compared to the traditional GARCH model. The prediction were plotted alongside the actual values and shown in Figure 4.20 and 4.21.

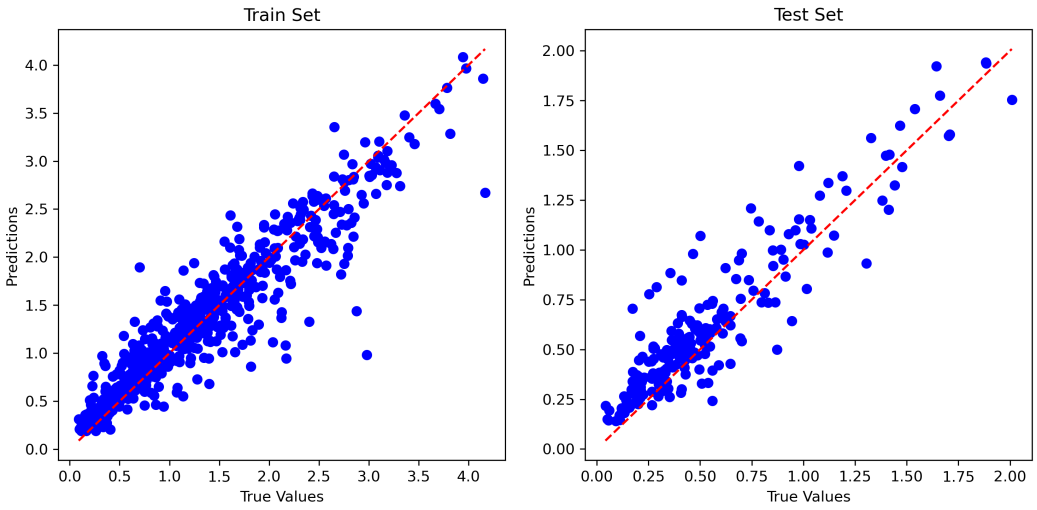

Figure 4.20. Train and test set of bitcoin volatility using LSTM model

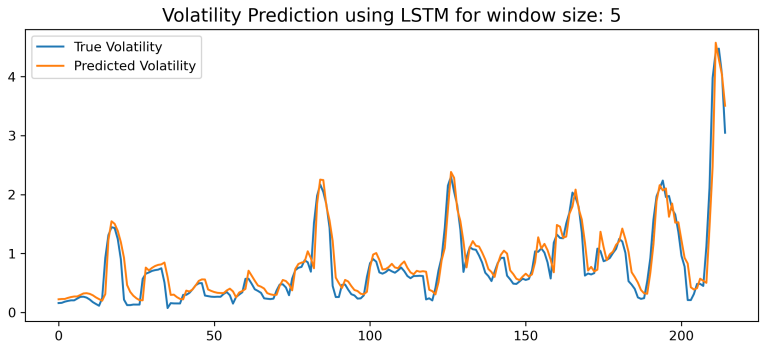

Figure 4.21. Volatility prediction of bitcoin volatility using LSTM model

GRU and BiLSTM models were also trained with similar configurations to the LSTM model, with 64 units each trained for 200 epochs. The performance metrics of the models is shown in Table 4.10. As evidenced from the performance metrics, LSTM, GRU, and BiLSTM performed similarly with very close RMSE, and MAE values. The MAPE for BiLSTM was the lowest, while the R squared for GRU was the highest.

Table 4.10. Comparison of test performances for volatility forecasting

	GARCH	LSTM	GRU	BiLSTM
RMSE	0.5090	0.2359	0.2305	0.2356
MAE	0.3484	0.1602	0.1535	0.1564
MAPE	157.5	35.88	35.88	32.32
R-squared	0.526	0.894	0.899	0.895

4.6. INTERACTIVE DASHBOARD

A user friendly dashboard is created as part of the thesis. In the dashboard, users can upload their dataset and perform predictive analysis using the various defined time series and machine learning models. The following models are available for analysis:

- Time series Models: ARCH and GARCH.

- Machine Learning Models: Random Forest, Gradient Boosting, Support Vector Regression, and Lasso Regression.
- Deep Learning Models: LSTM, GRU, BiLSTM.

The user can select certain parameters before training, which are describes as follows:

- Model Selection: Any model defined earlier can be selected for analysis.
- Range of dataset: Any range of dates can be selected as desired. In most of the analysis discussed here, the range of dataset selected was between 08/05/2020 and 03/05/2024.
- First Layer Units (or Neurons): This parameter is only applicable for the deep learning models, where the user can select the number of first layer units in their model.
- Second Layer Units (or Neurons): This parameter is also only applicable for deep learning models. By default, the number of second layer units is set to 0.
- Number of epochs: The number of epochs is the number of times the deep learning model is trained against the validation data.
- Window size: It is the number of past observations to consider for forecasting, and is used for all the models defined.
- Forecast Period (Days): The forecast period represents the number of days ahead for which the prediction is made. The prediction power of the defined models goes down as the forecast period increases.

The home page of the dashboard is shown in Figure 4.22.

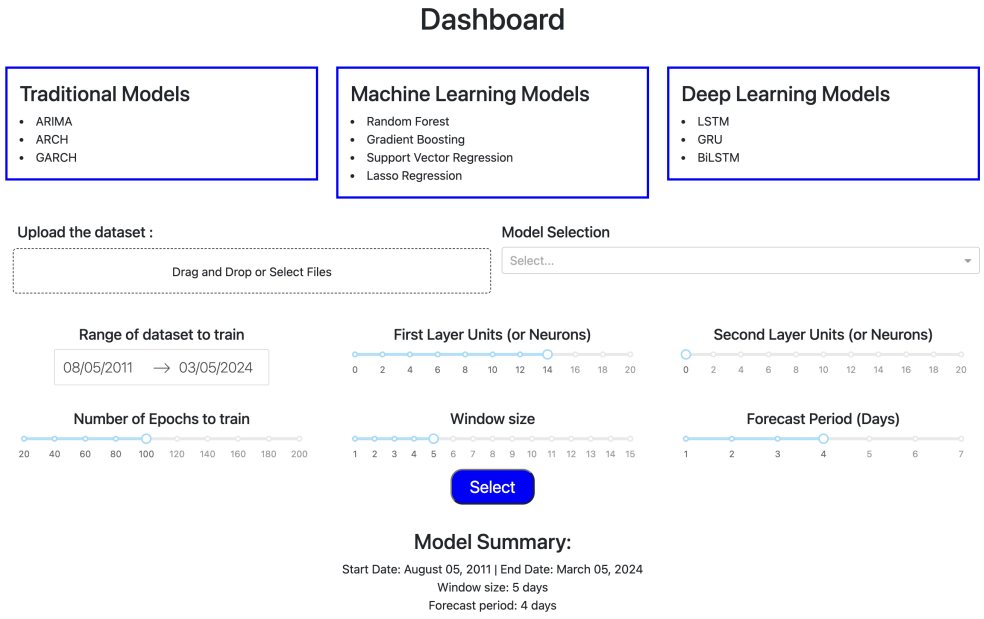

Figure 4.22. Interactive Dashboard for model training and evaluation

5. CONCLUSIONS

This thesis presented a comparative study of crypto volatility and price forecasting, employing a mixture of time series and machine learning models, including GARCH, LSTM, GRU, BiLSTM, and their hybrids as defined in [4], [10], [12]. The research aimed at exploring the effectiveness of these models in forecasting the volatility and price movements of cryptocurrencies, specifically focusing on their application to Bitcoin data. Through empirical analysis, it was demonstrated that different models have varying degrees of effectiveness, depending on the nature of the data and the specific forecasting requirements.

The development of an interactive dashboard for model training and evaluation represents a significant contribution to the field, enabling users to customize model parameters, train models, and compare their performance through an intuitive interface. This tool democratizes access to advanced forecasting techniques, making them accessible to a broader audience without requiring in-depth technical expertise.

Key Findings:

- Hybrid models obtained by combining various machine learning and time series methodologies showed promising results, indicating that this combination approach might be beneficial in capturing the complex dynamics of cryptocurrency markets.
- The user-friendly dashboard facilitated the empirical analysis by providing a platform for building sophisticated financial models.

Implications:

- The findings suggest that financial analysts and investors could significantly benefit from adopting a hybrid modeling approach for volatility and price forecasting in the highly volatile cryptocurrency market.

- The dashboard serves as a practical tool for real-time data analysis, model training, and evaluation, offering potential applications in educational settings and financial institutions.

Future Works:

- Further exploration of hybrid models with more diverse combinations of machine learning and time series methodologies could uncover more effective forecasting strategies.
- Extending the dashboard's functionality to include real-time data feeds and incorporating more financial assets (stocks and options) could enhance its utility and applicability.
- Expanding the impact of additional features, such as macroeconomic data or market sentiment data, on model performance could provide deeper insights into the predictive capabilities of the models.

In conclusion, this thesis reinforces the findings obtained by [4], [10], and [12], and introduces an innovative tool for model evaluation. The findings underscore the importance of continuous exploration and adaptation of forecasting tools to keep pace with the rapidly evolving financial landscape.

REFERENCES

[1] Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, pages 987–1007, 1982.

[2] William F Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The journal of finance*, 19.3:425–442, 1964.

[3] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31.3:307–327, 1986.

[4] Gustavo Di-Giorgi et al. Volatility forecasting using deep recurrent neural networks as garch models. *Computational Statistics*, Apr 2023.

[5] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58.1:267–288, 1996.

[6] Leo Breiman. Random forests. *Machine learning*, pages 5–32, 2001.

[7] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9.8:1735–1780, 1997.

[9] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45.11:2673–2681, 1997.

[10] Ze Shen et al. Bitcoin return volatility forecasting: A comparative study between garch and rnn. *Journal of Risk and Financial Management*, 14: 337, 2014.

[11] Hum Nath Bhandari et al. Predicting stock market index using lstm. *Machine Learning with Applications*, 9:100320, May 2022.

[12] Monghwan Seo and Geonwoo Kim. Hybrid forecasting models based on the neural networks for the volatility of bitcoin. *Applied Sciences*, 10(14):4768, July 2022.

[13] Jonathan D Cryer and Kung-Sik Chan. Time series analysis with applications in r. *Springer Science+Business Media, LLC*, 2008.

[14] Gareth James et al. An introduction to statistical learning: with applications in r. 2nd ed. *Springer Science+Business Media, LLC*, 2021.

[15] Robert E Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, pages 149–171, 2003.

[16] Grant Sanderson and Josh Pullen. Backpropagation calculus. *3Blue1Brown*, Nov 2017.

[17] Ian Goodfellow et al. Deep learning. *Cambridge, Massachusetts, The Mit Press*, 2016.

[18] Michael A. Nielsen. Neural networks and deep learning. *Determination Press*, 2015.

[19] Christopher Olah. Understanding lstm networks. *Colah's blog*, Aug 2015.

[20] K. Cho et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

VITA

Abhishek Kafle received his master's degree in Applied Mathematics with Statistics emphasis from Missouri University of Science and Technology. Prior to this, he earned his bachelor's degree in Mechanical Engineering from Stony Brook University in 2022.

During his academic journey, Abhishek worked as a teaching assistant, a research assistant, and an academic tutor. As a teaching assistant, he assisted in courses such as Applied Engineering Statistics and Multi variable calculus as a grader and a lab instructor. His passion for research led him to actively contribute to the paper titled "Optimal Control of Several Motion Models", which is published in the archives.