Predicting directions of S&P 500 using AI & ML models

by

Suraj Sakaria

A thesis submitted for the degree of

Masters in Philosophy

Department of Mathematics

Brunel University London

September 29, 2024

DECLARATION

I certify that this thesis, and the research to which it refers, are the product of my own work, and that any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Suraj Sakaria	Date
Dr. Cormac Lucas (Supervisor)	Date

Acknowledgements

First and foremost, I am grateful for the blessings and guidance of God throughout this academic journey. His presence has been a constant source of strength and inspiration.

I would like to express my sincere gratitude to OptiRisk Systems Ltd. for their generous sponsorship of my research. Their support has been invaluable in making this work possible.

In memory of Professor Gautam Mitra, who was instrumental in shaping the course of my studies and research. As the Director of OptiRisk, his vision and guidance were fundamental to this project. It is with profound honor and a sense of responsibility that I acknowledge my position as Professor Mitra's last sponsored student. His legacy continues to inspire and motivate our work, and I am deeply committed to carrying forward the torch of his academic vision.

I am deeply grateful to the entire OptiRisk team, especially Zryan Sadik and Arkaja Chakraverthy, whose insights and contributions were crucial to the development of this thesis.

My heartfelt thanks go to my supervisor, Dr. Cormac Lucas, for his unwavering support, regular meetings, and invaluable guidance throughout this journey. His mentorship has been instrumental in shaping both this thesis and my growth as a researcher.I would also like to extend my appreciation to Dr.Diana Roman for her valuable input and support during the course of this research. The Mathematical Sciences Department has been a constant source of support, and I am particularly grateful to Professor Paresh Date for his readiness to help during challenging times.

Lastly, I would like to acknowledge the unconditional support of my parents, family and

friends. Their encouragement, patience, and understanding have been the bedrock of my perseverance throughout this academic journey.

To all those mentioned and the many others who have contributed in various ways to this work, I offer my sincere thanks. This thesis would not have been possible without your collective support and guidance.

Abstract

The challenge of predicting (financial) market movements has long been based on quantitative models. These models are based on statistical analysis of historical data, which is used to identify patterns and relationships that are used to predict market behavior. Artificial intelligence (AI) and machine learning (ML) models are now extensively researched and applied in the domain of Financial Markets. The development of Python packages that encapsulate the theoretical underpinnings of these models has also taken place at the same time. Many teams in the finance industry have mastered and exploited these packages and report promising results in asset allocation and risk control. In our investigation of applying AI/ML and quant models we have used Python packages for the software realization of Random Forest, Gaussian Naïve Bayes, Decision Tree, Artificial Neural Net (ANN) and Logistic Regression models. The central aim of our investigation has been to generate next day market movement predictions specifically for S&P 500. The study employs a comprehensive dataset spanning five years, from 2017 to 2022, comprising closing prices of the S&P 500 index, VIX index, Gold, Oil, and 10-year U.S. Treasury yields as features. Technical indicators derived from these asset prices are utilized as additional features to enhance the predictive power of the models. Initially, the individual models are evaluated, with accuracy scores ranging from 50.90% (Random Forest) to 53.42% (Gaussian Naive Bayes), highlighting the inherent challenges of stock market prediction. To improve prediction performance, an ensemble modeling approach is adopted, where the final prediction is based on the majority voting of the

individual models. The results demonstrate that the ensemble modeling technique generally improves prediction accuracy, with the highest overall accuracy of 58.90% achieved when all models agree on the market direction. While the ensemble modeling approach shows promise, the achieved accuracy levels emphasize the complexity of stock market prediction tasks and the need for further enhancements. This research serves as a foundation for developing stock trading strategies by leveraging the ensemble model predictions for directional trading, risk management, portfolio optimization, and other comprehensive trading strategies.

Contents

Chapter 1

Introduction

1.1 Overview

The prediction of the stock market has long been a challenging field for traders and investors due to its inherent complexity. Various studies have been developed to predict stock market movements and improve prediction decisions. The early stages of predictive analysis were marked by pioneers who employed rudimentary time series analysis to discern patterns, trends, and cyclic movements within stock prices which, popularly came to be known as technical analysis. A notable advantage of technical analysis lies in its emphasis on objective data, such as stock prices and volumes, effectively eliminating subjective elements from the equation Murphy (1999). As technology advanced in the late 20th century, AI and ML began to redefine the landscape of stock market prediction. The ensuing decades witnessed the rise of algorithmic trading systems, underpinned by quantitative models, AI& ML algorithms and fuzzy logic, contributing to a new era of data-driven decision-making. Quantitative hedge

funds seized upon the power of AI and ML, deploying complex algorithms to harness vast datasets and uncover market inefficiencies. The integration of machine learning algorithms, ranging from support vector machines to deep learning architectures like convolutional and recurrent neural networks, further propelled predictive accuracy and analysis of time series data. However, it's important to acknowledge that while AI and ML have brought advances, the vast interlink of global events, psychological factors, and macroeconomic trends continue to challenge the accuracy of stock market predictions. As we start embracing predictive analysis, it is apparent that successful predictions requires integration of technology, quantitative prowess, and human expertise.

The advancement of statistical models for stock price prediction has been noteworthy. Basic regression methods establish a linear association between independent variables and make predictions about future values based on the analysis of historical data. However, studies have proven that stock prices exhibit non-linear behavior, and employing simple linear regression for forecasting may lead to inaccurate predictions Sarantis (2001). To overcome these limitations, alternative methodologies such as the Auto-Regressive Integrated Moving Average (ARIMA) model have gained prominence in the prediction of stock prices and market trends and research by Ariyo et al. (2014) has shown that the ARIMA model has a strong potential for short-term prediction. The ARIMA model combines autoregression, moving average, and differencing techniques to capture both linear relationships and patterns present in stock price data. However, due to the complex, highly dynamic, and chaotic nature of the stock market, time series models have limitations in capturing these complex relationships. Furthermore, stock market data is known for its high noise levels, making it

challenging for time series models to accurately predict future prices.

The rapid growth in computing power has led to increased use of artificial intelligence and machine learning techniques for stock market prediction. These techniques have gained popularity because they are able to capture complex relationships within stock market data and can combine multiple indicators to make predictions. By utilizing large quantities of historical data and employing sophisticated algorithms, these methods can reveal patterns and trends that traditional time-series models find challenging to identify. With the expanding availability of computational resources, researchers and analysts now have the ability to employ more advanced artificial intelligence and machine learning algorithms, including deep learning models which excel at detecting intricate patterns from vast volumes of structured and unstructured data. They can also adapt and learn from complex datasets using various algorithms, unlike conventional statistical models that make assumptions about data distributions. Furthermore, advances in computational capabilities have also facilitated faster processing times, enabling the analysis of large datasets with enhanced precision and efficiency.

Machine learning models such as Logistic Regression have gained popularity in the field of stock market prediction due to its effectiveness and versatility. Logistic regression is a supervised learning algorithm that is well-suited for classification problems. One example is a study by Dutta et al. (2012) to classify the companies with up to a 74.6% level of accuracy into two categories-" good" or "poor", based on their rate of return. Similarly, Decision tree and naive Bayes classifier are two other widely used machine learning techniques for stock market prediction. Decision Tree is a supervised learning algorithm known for

its ability to capture intricate relationships within the data by constructing a decision-based model represented as a tree. Research by Fiévet and Sornette (2018) shows that Decision tree models were found to significantly outperform linear autoregressive models in predicting daily returns of the S&P 500 over a 20-year period, particularly during market crashes, by capturing non-linear interactions between lagged directional price movements that linear models miss. On the other hand, Naive Bayes classifier is a probabilistic algorithm that assumes conditional independence among features given class labels. This simplifying assumption allows it to efficiently analyze large datasets and make predictions based on probability calculations. An example of Naive Bayes prediction model is by Khedr et al. (2017) where the researchers used sentiment analysis of financial news and historical stock market prices to make predictions. Additionally, neural networks have gained prominence as powerful tools in forecasting stock prices due to their capacity in handling complex patterns and capturing non-linear dependencies. In recent studies, it has been found that neural networks like Convolutional Neural Network, Multilayer Perceptrons and Long Short-Term Memory Networks have demonstrated significant potential in the prediction of stock market trends Hiransha et al. (2018). These models, particularly multilayer perceptrons, have been utilized to create robust frameworks that enhance the accuracy of forecasting the closing index price performance of influential indices such as The Dow Jones Industrial Average, Johannesburg Stock Exchange All Share, Nasdaq 100 and the Nikkei 225 Stock Average index Patel and Marwala (2006).

Furthermore, different machine learning algorithms can be used to improve the accuracy of stock market forecasting through ensemble techniques such as Random Forests and Gra-

dient Boosting which are effective methods that combine predictions from multiple models. These ensemble methods use a collection of base models, each with their own strengths and weaknesses, to generate more accurate forecasts. Ballings et al. (2015) benchmarked different ensemble methods (Random Forest, AdaBoost and Kernel Factory) against single classifier models (Neural Networks, Logistic Regression, Support Vector Machines and K-Nearest Neighbor) and concluded that Random Forest is the top algorithm followed by Support Vector Machines, Kernel Factory, AdaBoost, Neural Networks, K-Nearest Neighbors and Logistic Regression.

Another advantage of AI & ML models is that the flexibility in selecting input features is higher compared to traditional stock price forecasting models such as Multiple Linear Regression, ARCH, GARCH, ARIMA etc. which utilize a limited set of input features. The importance of technical indicators as input features lies in its ability to capture short-term price fluctuations and identify potential entry and exit points for trading Shynkevich et al. (2017). The VIX, also known as the CBOE Volatility Index, is an important feature for stock prediction that can be used in machine learning models. The VIX is a measure of market volatility and is often referred to as the "fear gauge" as it reflects investors' sentiment towards the market, with researchers such as Smales (2017) using it as a proxy for market sentiment. Overall, the VIX has proven to be a useful tool in making stock market predictions. Study by Wang (2019) shows that the VIX's inclusion in stock market prediction models can improve their forecast accuracy. Similarly, other variables like Gold, crude oil, and interest rates are also commonly used as features in stock market prediction models. These features provide valuable insights into the macroeconomic factors that may impact stock prices Mensi et al.

(2013). Gold is often considered a safe-haven asset and tends to attract investors during times of economic uncertainty. Thus, changes in gold prices can provide information about market sentiment and risk appetite. Crude oil prices, on the other hand, are closely tied to the global economy and can indicate the state of economic growth and demand and research has shown that changes in oil prices can have a significant impact on stock market performance Kilian and Park (2009). Interest rates, set by central banks, can have a significant impact on investment decisions and borrowing costs, thereby influencing stock market performance as researched by Campbell (1987). By incorporating these variables as features, machine learning models can capture the underlying relationships between the stock market and these macroeconomic factors, thereby enhancing the accuracy of stock market predictions and providing a more comprehensive understanding of the dynamics and trends within the market.

In recent years, there has been growing interest in incorporating sentiment data as a feature for stock prediction. Sentiment data refers to the information gathered from various sources, such as news articles, social media posts, and financial reports, that reflects investors' emotions and attitudes towards the market. This type of data can provide valuable insights into market sentiment and machine learning techniques can be used to process the textual input of news stories to determine quantitative sentiment scores Mitra and Mitra (2011) . Research by Deng et al. (2011) have proven that by incorporating sentiment data as a feature in machine learning models can capture the impact of investors' sentiment on stock prices and improve the accuracy of prediction.

Finally, it is important to note that classification tasks in finance, particularly predicting

market direction, often yield surprisingly low accuracy rates, a fact that may not be widely appreciated outside of specialized circles. For instance, Leung et al. (2000) achieved only 54% accuracy in predicting the direction of the NIKKEI 225 index using neural networks. Similarly, Kara et al. (2011) reported accuracy rates of 75.74% and 71.52% using artificial neural networks and support vector machines respectively on the Istanbul Stock Exchange using top 10 technical indicators as features, which, while better, still leave significant room for error. Even more sophisticated approaches, such as the deep learning model employed by Chong et al. (2017), only achieved an accuracy of 65.1% in predicting the Korea Composite Stock Price Index. These results underscore the inherent difficulty of the task, which stems from the complex, non-linear nature of financial markets and the multitude of factors influencing price movements. As pointed out by Gu et al. (2020) in their comprehensive review, even state-of-the-art machine learning methods struggle to consistently outperform simple benchmarks in financial prediction tasks. This persistent challenge highlights the need for continued research into novel features, advanced modeling techniques, and innovative approaches to potentially improve prediction accuracy.

This research paper presents an extensive computational study to investigate the effectiveness of machine learning (ML) methods in predicting the directional movement of the S&P 500 index. The primary objective is to answer the research question: Can ML techniques accurately forecast the daily direction (up or down) of the S&P 500? The study employs a comprehensive dataset spanning five years, from 2017 to 2022, comprising closing prices of the S&P 500 index, VIX index, Gold, Oil, and 10-year U.S. Treasury yields. These variables serve as the independent features for the ML models. Additionally, 21 different

technical indicators derived from these asset prices are utilized as supplementary features to enhance the predictive power of the models. The research compares the performance of individual ML models, including Logistic Regression, Decision Tree, Random Forest, Gaussian Naive Bayes, and Multilayer Perceptron (MLP), and further explores the potential of ensemble modeling techniques to improve prediction accuracy.

1.2 Organisation of Thesis

This report is structured as follows:

Section 2 Methodology: Delves into the mathematical formulations and theories behind the various machine learning models employed, including Logistic Regression, Decision Trees, Random Forests, Gaussian Naive Bayes, and Multi-Layer Perceptrons (Neural Networks). It provides detailed architectural specifications for each model, outlining the exact structure and parameters used. This section also explains the ensemble modeling approach of combining predictions from multiple models to improve overall accuracy.

Section 3 Data: Outlines the sources of market data (S&P 500, VIX, gold, oil, Treasury yields) and the calculation of returns. It provides a comprehensive list of the 21 technical indicators used as features, along with their descriptions, formulas, and potential implications for market analysis. The section also justifies the use of multiple indicators based on their ability to capture various aspects of market behavior.

Section 4 Empirical Investigation: Describes the experimental setup using Python and scikit-learn, including the rolling window approach for training and testing. It presents the results and analysis of individual model performances using default configurations, as well

as the hyperparameter tuning process for the MLP model. This section includes a detailed analysis of the optimized MLP configuration, comparing its performance to the default settings. It also presents the results of the ensemble model under different voting scenarios and discusses the observed discrepancies in upward/downward prediction accuracies.

Section 5 Conclusion and Discussion: Summarizes key findings, including the performance of individual models, the impact of hyperparameter tuning on the MLP model, and the results of the ensemble approach. It discusses the challenges encountered in the study, such as the limitations of the rolling window approach for model evaluation. The section outlines future research directions, including potential improvements through further hyperparameter tuning of other models, addressing prediction asymmetries, and refining ensemble techniques. Finally, it explores potential applications of the research findings in developing stock trading strategies, including directional trading, risk management, and portfolio optimization.

Chapter 2

Methodology

2.1 Logistic Regression

In logistic regression, our goal is to model the relationship between a set of predictor variables (features) and a binary dependent variable (outcome). Specifically, we want to predict whether the S&P 500 index will open UP or DOWN the next day based on various market and economic indicators.

Given a training dataset consisting of n observations, where each observation i includes:

- A vector of p predictor variables (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$

  These features could include various market and economic indicators such as the previous day's closing price, VIX, economic indicators, and other relevant factors that may influence the S&P 500 index.

- A binary outcome: $y_i \in \{0, 1\}$

  In this case, y_i represents whether the S&P 500 index opened UP ($y_i = 1$) or DOWN

($y_i = 0$) on the corresponding day.

Our task is to estimate the parameters (coefficients) $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$ that define the logistic regression model. These coefficients represent the weights assigned to each predictor variable, indicating their influence on the probability of the S&P 500 index opening UP or DOWN. The goal is to find the coefficients that best fit the training data, minimizing the difference between the predicted probabilities and the actual outcomes.

Model Specification

Logistic regression is used to model the probability that a given observation belongs to a particular category Hosmer (2000). In binary logistic regression, this probability is modeled as a function of one or more features. Let the probability of the i-th observation belonging to the category of interest (the S&P 500 opening higher) as $\pi_i = P(Y_i = 1 \mid \mathbf{x}_i)$, where $\mathbf{x}_i$ is the vector of predictors for the i-th observation.

The Logistic Function

To model this probability π_i, the logistic function or sigmoid function is used:

$$\pi_i = \frac{1}{1 + e^{-g(\mathbf{x}_i)}} \tag{2.1}$$

where

- $\pi_i = P(Y_i = 1 \mid \mathbf{x}_i)$ is the probability that the i-th observation results in an event (e.g., the S&P 500 opening higher)

- $g(\mathbf{x}_i)$ is a linear combination of the predictors.

And for multiple predictors, it takes the form:

$$g(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \tag{2.2}$$

where,

- β_0 is the intercept
- β_j (for $j = 1, 2, \ldots, p$) are the coefficients for the predictors.

The importance of logistic function is that it converts $g(\mathbf{x}_i)$, which can take any real value - infinity to + infinity , into a value between 0 and 1. This transformation is necessary because probabilities cannot exceed this range.

The Odds and the Logit Function

In logistic regression, we also work with odds and log-odds (logits). This is because they provide a straightforward way to model and interpret the relationship between the features and the probability of an outcome.

The odds of the event occurring (e.g., the S&P 500 opening higher) is defined as the ratio of the probability of the event occurring to the probability of it not occurring:

$$\text{Odds} = \frac{\pi_i}{1 - \pi_i} \tag{2.3}$$

The logit function is the natural logarithm of the odds. We do this transformation because the logit function transforms the probability π_i into log-odds, which can take any

real value from - infinity to + infinity linearising the relationship between the features and the outcome probability, making it easier to model using linear regression techniques. Also, the coefficients in the model represent changes in the log-odds of the outcome, making it easier to understand the effect of each feature on the likelihood of the outcome.

$$\text{Logit}(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right) \tag{2.4}$$

Deriving the Logit Model

By substituting the logistic function into the logit function, we get:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \log\left(\frac{\frac{1}{1+e^{-g(\mathbf{x}_i)}}}{1-\frac{1}{1+e^{-g(\mathbf{x}_i)}}}\right)$$

Simplifying the expression inside the logarithm:

$$1-\pi_i = 1-\frac{1}{1+e^{-g(\mathbf{x}_i)}} = \frac{e^{-g(\mathbf{x}_i)}}{1+e^{-g(\mathbf{x}_i)}}$$

So,

$$\frac{\pi_i}{1-\pi_i} = \frac{\frac{1}{1+e^{-g(\mathbf{x}_i)}}}{\frac{e^{-g(\mathbf{x}_i)}}{1+e^{-g(\mathbf{x}_i)}}} = e^{g(\mathbf{x}_i)}$$

Taking the natural logarithm of both sides:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = g(\mathbf{x}_i)$$

Since $g(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}$, we have:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} \tag{2.5}$$

This equation is the fundamental form of the multiple logistic regression model, expressing the log-odds of the event occurring as a linear combination of the predictors.

Estimation of Parameters

Likelihood Function

The parameters of the logistic regression model are typically estimated using maximum likelihood estimation (MLE). It helps us find the best-fitting coefficients for our model that aligns closely with the observed data.

We use the likelihood function because it represents the probability of observing the given data as a function of the model parameters. The likelihood function combines the predicted probabilities π_i of each observation being 1 or 0 and multiplies them to reflect the overall likelihood of observing the given data with the current parameter values. By maximizing this function, we find the best-fitting parameters for our model.

Given n independent observations $(\mathbf{x}_i, y_i)$, the likelihood function $L(\beta)$ is the product of the probabilities of the observed outcomes:

$$L(\beta) = \prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i} \tag{2.6}$$

Where:

- $\pi_i^{y_i}$ represents the contribution to the likelihood function being 1 when $y_i = 1$.

- $(1-\pi_i)^{1-y_i}$ represents the contribution to the likelihood function being 0 when $y_i = 0$.

$\pi_i^{y_i}$ and $(1-\pi_i)^{1-y_i}$ reflect the probability of the observed outcome y_i for each observation i, and the product over all observations gives the overall likelihood of the observed data under the logistic regression model.

And this likelihood function represents the probability of obtaining the observed data given the parameters β.

Log-Likelihood Function

For convenience, we work with the log-likelihood function $\ell(\beta)$, which is the natural logarithm of the likelihood function. This is because maximizing the likelihood function directly can be cumbersome due to the complexity and instability of multiplying many small probability values, making numerical computations difficult and less stable. Thus, by converting the product into a sum using the natural logarithm of the likelihood function, we simplify the maximization process as given below,

$$\ell(\beta) = \log L(\beta) = \sum_{i=1}^{n} \left[y_i \log(\pi_i) + (1 - y_i) \log(1 - \pi_i) \right] \tag{2.7}$$

Substituting the Logistic Function

Substitute $\pi_i = \frac{1}{1+e^{-g(\mathbf{x}_i)}}$ into the log-likelihood function:

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i \log \left(\frac{1}{1 + e^{-g(\mathbf{x}_i)}} \right) + (1 - y_i) \log \left(\frac{e^{-g(\mathbf{x}_i)}}{1 + e^{-g(\mathbf{x}_i)}} \right) \right]$$

Simplify the terms inside the logarithms:

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i \left(-\log(1 + e^{-g(\mathbf{x}_i)}) \right) + (1 - y_i) \left(-g(\mathbf{x}_i) - \log(1 + e^{-g(\mathbf{x}_i)}) \right) \right]$$

Combine the terms:

$$\ell(\beta) = \sum_{i=1}^{n} \left[-y_i \log(1 + e^{-g(\mathbf{x}_i)}) - (1 - y_i) g(\mathbf{x}_i) - (1 - y_i) \log(1 + e^{-g(\mathbf{x}_i)}) \right]$$

$$\ell(\beta) = \sum_{i=1}^{n} \left[-y_i \log(1 + e^{-g(\mathbf{x}_i)}) - g(\mathbf{x}_i) + y_i g(\mathbf{x}_i) - \log(1 + e^{-g(\mathbf{x}_i)}) + y_i \log(1 + e^{-g(\mathbf{x}_i)}) \right]$$

$$\ell(\beta) = \sum_{i=1}^{n} \left[y_i g(\mathbf{x}_i) - \log(1 + e^{g(\mathbf{x}_i)}) \right] \tag{2.8}$$

Maximizing the Log-Likelihood Function

To find the estimates $\hat{\beta}$, we maximize the log-likelihood function $\ell(\beta)$. This involves taking the partial derivative of $\ell(\beta)$ with respect to each β_j (for $j = 0, 1, \ldots, p$) and setting it to zero.

Derivative of the Log-Likelihood

For β_j:

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \left[y_i \frac{\partial g(\mathbf{x}_i)}{\partial \beta_j} - \frac{e^{g(\mathbf{x}_i)}}{1 + e^{g(\mathbf{x}_i)}} \frac{\partial g(\mathbf{x}_i)}{\partial \beta_j} \right]$$

Since $g(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$, the derivative $\frac{\partial g(\mathbf{x}_i)}{\partial \beta_j} = x_{ij}$. Therefore,

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \left[y_i - \frac{e^{g(\mathbf{x}_i)}}{1 + e^{g(\mathbf{x}_i)}} \right] \quad \text{for } j = 0$$

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} \left[y_i x_{ij} - \frac{e^{g(\mathbf{x}_i)}}{1 + e^{g(\mathbf{x}_i)}} x_{ij} \right] \quad \text{for } j = 1, 2 \ldots, p$$

Simplify using $\pi_i = \frac{e^{g(\mathbf{x}_i)}}{1+e^{g(\mathbf{x}_i)}}$:

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - \pi_i) \quad \text{for } j = 0 \tag{2.9}$$

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} x_{ij}(y_i - \pi_i) \quad \text{for } j = 1, 2 \ldots, p \tag{2.10}$$

Finally, we set the partial derivatives equal to zero and solved using the gradient ascent :

$$\sum_{i=1}^{n} (y_i - \pi_i) = 0 \quad \text{for } j = 0 \tag{2.11}$$

$$\sum_{i=1}^{n} x_{ij}(y_i - \pi_i) = 0 \quad \text{for } j = 1, 2 \ldots, p \tag{2.12}$$

Solving Using Gradient Ascent (or Descent)

The Gradient ascent is an optimization algorithm used to maximize a function. In our context, it is used to maximize the log-likelihood function $\ell(\beta)$, thereby finding the parameter values β that best fit the data.

In gradient ascent, the parameters β are updated iteratively in the direction of the gradient of the log-likelihood function. This ensures that each step increases the log-likelihood, moving towards the maximum.

Gradient Ascent Algorithm

1. Initialization:

- Choose initial values for the parameters β, often starting with zeros or small random values.

- Set the learning rate α, which controls the step size of each update. The value of α is typically chosen through experimentation.

2. Compute the Gradient:

- Calculate the gradient of the log-likelihood function with respect to each parameter β_j. The gradient (or score function) for β_j is given by:

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - \pi_i) \quad \text{for } j = 0$$

$$\frac{\partial \ell(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} x_{ij}(y_i - \pi_i) \quad \text{for } j = 1, 2 \ldots, p$$

where $\pi_i = \frac{1}{1+e^{-g(\mathbf{x}_i)}}$ and $g(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$.

3. Update the Parameters:

- Update each parameter β_j using the gradient ascent rule:

$$\beta_j^{(t+1)} = \beta_j^{(t)} + \alpha \frac{\partial \ell(\beta)}{\partial \beta_j} \quad \text{for } j = 0, 1, 2 \ldots, p \tag{2.13}$$

where $\beta_j^{(t)}$ is the value of β_j at iteration t.

4. Iterate Until Convergence:

- Repeat steps 2 and 3 until convergence. Convergence is typically determined by checking if the change in the log-likelihood or the parameter values between iterations is smaller than a predefined threshold.

Thus, by iteratively updating the parameters in the direction of the gradient, we ensure that each step moves us closer to the optimal parameter values. The learning rate α is crucial for controlling the size of each update, and it must be chosen carefully to balance the speed of convergence and the stability of the optimization process.

This process ensures that the estimated parameters $\hat{\beta}$ maximize the probability of observing the given data under the logistic regression model.

Once we have estimated the coefficients, we can use the logistic regression model to make predictions on new, unseen data. By inputting the values of the predictor variables for a new day, the model will output the probability of the S&P 500 index opening UP or DOWN.

2.2 Decision Tree

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. A decision tree starts at the top and works its way down until it gets to a point where it can't classify any further. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes, and leaf nodes.

From the diagram above, a decision tree starts with a root node which does not have any incoming branches. The outgoing branches from the root node then feed into the internal

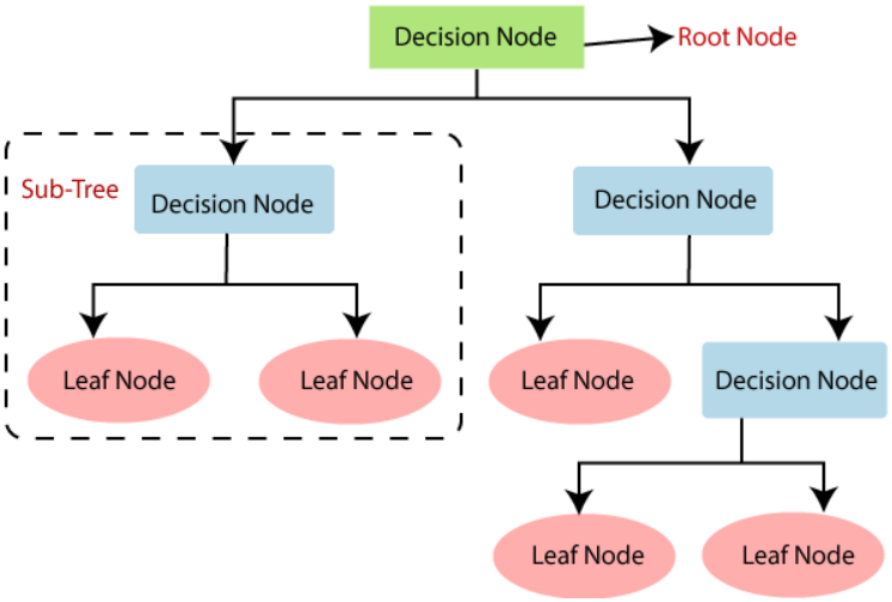

Figure 2.1: Decision Tree

nodes, also known as decision nodes. Based on the available features, both node types conduct evaluations to form homogeneous subsets, which are denoted by leaf nodes or terminal nodes. The leaf nodes represent all possible outcomes within the data set.

The most important step in creating a decision tree, is the splitting of the data. The decision tree has to find a way to split the data set (D) into two data sets (D1) and (D2). There are different ways that can be used to find the next split. The Gini Impurity method of splitting Strobl et al. (2007), which is a criterion for categorical target variables is one of the most common methods practiced and is also the criterion used by the Python library scikit-learn.

Gini impurity is a measure that is used to determine how mixed a prediction classification is. If all items in the classification are of the same type, eg: all classification is 'UP' in a stock market prediction, the Gini impurity is 0, meaning the set is pure. If the set is evenly split

between different types, eg: 'UP' and 'DOWN' the Gini impurity is 0.5, indicating maximum impurity. Essentially, it tells us the probability of misclassifying a randomly chosen set of items in a set.

The following steps show how decision tree uses Gini Impurity for splitting.

1. Calculate the Gini Impurity for the current dataset. This is the impurity before any split.

2. Evaluate Potential Splits.

For each feature, evaluate the possible splits which involves systematically testing all potential ways to divide the dataset based on each feature. For a numeric feature, this involves considering all possible threshold values that can split the dataset into two parts. For a categorical feature, it involves considering all possible ways to divide the categories into two groups.

3. Calculate Gini Impurity for Each Split

For each potential split:

- Split the dataset into two subsets.
- Calculate the Gini impurity for each subset.
- Calculate the weighted average of these impurities to get the impurity of the split.

The Gini impurity of a split can be calculated as:

$$\text{Ginni for each subset} = 1 - (\text{Probability of Yes})^2 - (\text{Probability of No})^2 \tag{2.14}$$

Gini Impurity = Weighted average of the Gini at each leaf.

$$\text{Gini Impurity}(D) = \frac{n_1}{n} \cdot \text{Gini}(D_1) + \frac{n_2}{n} \cdot \text{Gini}(D_2) \tag{2.15}$$

with n = $n_1 + n_2$ the size of the data set (D)

4. Choose the Split with the Lowest Gini Impurity The split that results in the lowest weighted Gini impurity is chosen because it best separates the data into pure subsets.

So to build a desision tree, the first thing we have to decide is, which feature is going to be the root node. To identify the root node feature, we evaluate all features by testing their possible splits, calculating the resulting impurities, and selecting the feature that provides the split with the lowest weighted average impurity.

Keeping this root node, the process is repeated for the remaining features to find the one that has the lowest Gini impurity. This becomes the variable for the next node.

We continue this splitting until a stopping criterion is met, which are:

- All data points in a subset belong to the same class.
- A maximum tree depth is reached.
- A minimum number of data points in a node is reached.
- Further splitting does not improve impurity significantly.

In a decision tree, splits are applied recursively. This means the process of finding the best split and dividing the data continues repeatedly for each subset until the decision tree is fully grown according to the stopping criteria. At each step, the algorithm selects the

optimal feature and threshold to split the data, creating smaller and more homogeneous subsets, until it can no longer make meaningful splits.

Finally, a decision tree structure is built with:

- **Root Node**: The initial dataset.
- **Internal Nodes**: Each node represents a split based on a feature and a threshold.
- **Leaf Nodes**: The terminal nodes where further splitting is not performed. Each leaf node represents a class label.

For stock market prediction in our research, a decision tree can predict the direction of movements by recursively splitting historical data based on features like the previous day's price, VIX, and other indicators by minimizing impurity using Gini impurity. At each node, the best feature and threshold for splitting the data are chosen to create more homogeneous subsets. This process continues until stopping criteria are met, forming a tree with leaf nodes representing final predictions i.e market going 'UP' or 'DOWN'. For new data, predictions are made by following a path from the top of the decision tree (the root) down to one of the final nodes (leaf nodes) by making decisions at each internal node based on the feature values of the data point being classified.

Decision trees are intuitive, easy to understand and interpret and the data doesn't need to be scaled. However, overfitting is a common problem with decision trees Bramer (2007). We can over come this by 1) Pruning and 2) Set a limit on how the tree grows. The Sklearn Python package helps with setting up those limits.

2.3 Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to make accurate predictions in classification tasks. Ensemble learning involves training multiple individual models and combining their predictions to obtain a final prediction. Though decision trees are simple to construct, one of the major drawbacks of decision tree as a predictive model is their inaccuracy of classifying new data. Random Forest combines the simplicity of decision tree with flexibility, resulting in vast improvement in accuracy Breiman (2001). Random forest starts with the process of Bootstrapping. The Bootstrapping process has the following steps,

1) Start with an original dataset of size N, containing observations or samples.

2) Randomly select an observation from the original dataset and add it to the bootstrapped sample.

3) Put the selected observation back into the original dataset, allowing it to be selected again.

4) Repeat steps 2 and 3 until the bootstrapped sample reaches the desired size (usually the same as the original dataset).

5) This process of randomly selecting with replacement creates a new dataset that may contain duplicate observations.

6) Repeat steps 2-5 to generate multiple bootstrapped datasets, usually referred to as bootstrap replicates.

Now for each bootstrapped dataset a decision trees is created. The process of creating the decision trees is similar to what was explained earlier. However, in Random Forest at each

node a random subset of features is selected for splitting. Once the trees are constructed, the new data are run through the various decision trees created by the Random Forest. The results of the individual trees are then aggregated and the prediction that receives the highest votes is selected. In our case, each decision trees will predict if the market will go 'UP' or 'DOWN' the next day and the prediction results are aggregated. If the majority of the trees classify that the market will go 'UP' tomorrow then that would be the prediction, else the random forest prediction is that the market will go 'DOWN' the next day.

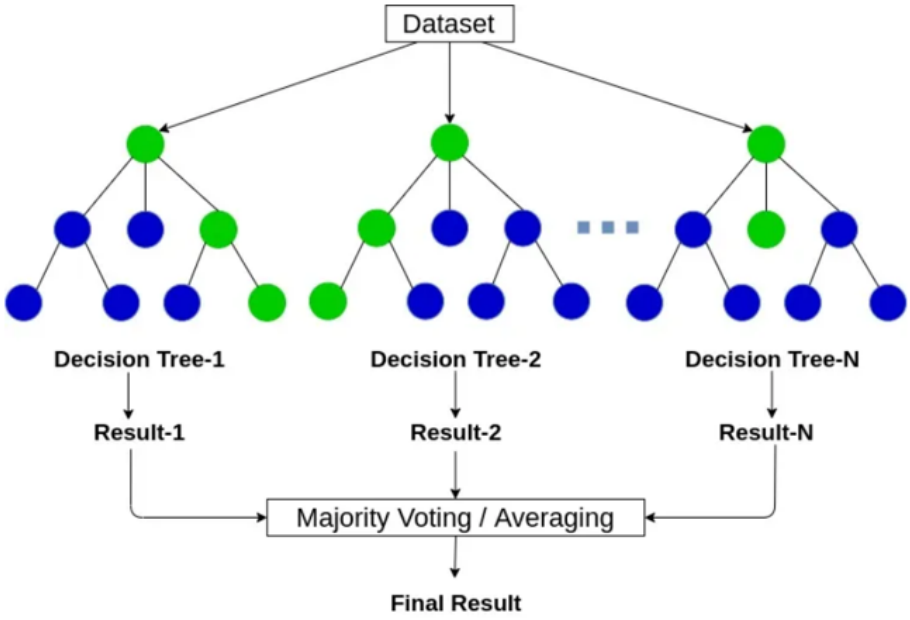

Figure 2.2: Random Forest

This process of Bootstrapping the data and then aggregating the results to make a decision is called "Bagging". This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias Altman and Krzywinski (2017). This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not

correlated.

Random Forest makes predictions by aggregating the predictions of individual trees in the ensemble. For classification tasks, the class with the highest number of votes among the trees is selected as the final prediction. The decision-making process involves traversing the trees based on the feature values of new instances. In contrast to the original publication , the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

2.4 Gaussian Naive Bayes

Gaussian Naive Bayes is an extension of the Naive Bayes classification algorithm, particularly useful for problems involving continuous numerical data. Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem, a simple mathematical formula used to calculate conditional probabilities Rish et al. (2001). Conditional probability measures the likelihood of an event occurring given that another event has already occurred.

Bayes' Theorem is expressed as:

$$P\left(A|B\right) = \frac{P\left(B|A\right) \cdot P\left(A\right)}{P\left(B\right)} \tag{2.16}$$

Where:

- $P(A|B)$ is the probability of event A occurring given that event B has occurred (posterior probability).

- $P(B|A)$ is the probability of event B occurring given that event A has occurred.

- $P(A)$ is the probability of event A occurring.

- $P(B)$ is the probability of event B occurring.

Naive Bayes makes two fundamental assumptions about the features:

1. Independence : Each feature is independent of the others.

2. Equal Contribution : Each feature contributes equally to the outcome.

For example, consider features like 'VIX' (Volatility Index) and 'Gold Futures'. Naive Bayes assumes:

- The 'VIX' is independent of 'Gold Futures'.

- Both 'VIX' and 'Gold Futures' contribute equally to the prediction outcome.

These assumptions made by Naıve Bayes are generally not correct in real-world situations. The independence assumption is never correct, but it has been found to work well in practice. Hence the name Naive. Because Naive Bayes ignores the relationship amongst the features, it is said to have high bias. However, since it works well in practice, they have low variance.

The Bayes' theorem for classification can be rewritten as:

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)} \tag{2.17}$$

Where y is the class variable and X represents the features:

$$X = (x_1, x_2, x_3, \ldots, x_n) \tag{2.18}$$

For Naive Bayes, this expands to:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(x_1 \mid y)P(x_2 \mid y) \cdots P(x_n \mid y)P(y)}{P(x_1)P(x_2) \cdots P(x_n)} \tag{2.19}$$

The denominator $P(x_1)P(x_2) \cdots P(x_n)$ remains constant for all entries in the dataset, allowing us to simplify the expression to proportionality:

$$P(y|x_1 \ldots x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y) \tag{2.20}$$

To make predictions, we calculate the posterior probability $P(y|X)$ for each class and select the class with the highest posterior probability.

However, Naive Bayes has a limitation known as the "zero-frequency problem." If a class label and a specific attribute value never occur together in the training data, the probability estimate will be zero. To address this, we can add one to the count for every attribute value-class combination (Laplace smoothing).

For continuous numerical features, Gaussian Naive Bayes assumes a normal (Gaussian) distribution. The probability density function for a normal distribution is:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.21}$$

Where:

- μ is the mean:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.22}$$

- σ is the standard deviation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.23}$$

With these parameters, we can calculate the likelihood of the market going 'UP' or 'DOWN' given new feature values for the next day. The classifier will predict the class ('UP' or 'DOWN') with the highest posterior probability.

As an example to understand the model, let us consider a case where we aim to predict the next day direction (Up/Down) of the SP500 using VIX. The predictor or the dependent variable is the market direction, and the independent variable i.e feature is the VIX.

The below tabel consisits of 10 data points from 16/11/2022 to 30/11/2022 of the SP500 and VIX which we shall use to explain the mathematical algorithms behind various models.

Date	Next Day SP500 movement (Up/Down) (y_t)	VIX (x_1)	SP 500 today close (x_2)	VIX Close today (x_3)
30/11/2022	1	20.58	1	0
29/11/2022	0	21.89	1	1
28/11/2022	1	22.21	0	1
25/11/2022	1	20.5	1	1
23/11/2022	1	20.42	1	1
22/11/2022	1	20.35	1	0
21/11/2022	0	21.29	1	1
18/11/2022	1	19.43	0	0
17/11/2022	0	23.12	1	0
16/11/2022	0	23.93	0	0

Figure 2.3: Dataset

y_t = Next day movement of SP 500 index. (Up / Down)

t = Time period

x_1, x_2, x_3 = Features that are used to predict the market direction.

Where,'1' is 'UP' and '0' is 'DOWN'

From the dataset, let us create frequency and likelihood tables the categorical feature 'SP 500 today close'(x_2) and 'VIX Close today' (x_3).

Frequency Table

	Next Day SP500 movement?		
		Up	Down
SP500 today close	1	4	3
SP500 today close	0	2	1

Table 2.1: Frequency Table - SP 500 today close

Likelihood Table

	Next Day SP500 movement?		
		Up	Down
SP500 today close	1	4/6	3/4
SP500 today close	0	2/6	1/4

Table 2.2: Likelihood Table - SP 500 today close

However, for VIX, which contains numerical value, one option is to transform the numerical values to their categorical counterparts before creating their frequency tables. The other option, as shown below, could be using the distribution of the numerical variable to have a good guess of the frequency. One common method is to assume normal or Gaussian

Frequency Table

	Next Day SP500 movement?		
		Up	Down
VIX Close today	1	3	2
VIX Close today	0	3	2

Table 2.3: Frequency Table - VIX Close today

Likelihood Table

	Next Day SP500 movement?		
		Up	Down
VIX Close today	1	3/6	2/4
VIX Close today	0	3/6	2/4

Table 2.4: Likelihood Table - VIX Close today

distributions for numerical variables and hence the term Gaussian Naïve Bayes. The probability density function for the normal distribution is defined by two parameters, mean and standard deviation. In Gaussian Naïve Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution i.e Normal distribution.

The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.24}$$

Where,

$$\text{Mean } (\mu) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.25}$$

$$\text{Standard Deviation } (\sigma) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.26}$$

So, with the given values of VIX the mean and standard deviation are calculated and summarised below.

Table 2.5: Frequency / Likelihood Table

Next Day SP500 movement?	**1**	**0**
Vix	19.43, 20.35, 20.42, 20.5, 22.21, 20.58	23.93, 23.12, 21.29, 21.89
Mean	20.58	22.56
Std. Deviation	0.82	1.19

Once we have this, we can calculate the likelihood of the market going Up or Down for the next day, (i.e, 01/12/2022). The features recorded on 30/12/2022 are,

Table 2.6: Data Table (30/12/2022)

VIX (X1)	**S&P 500 Today Close (X2)**	**VIX Close Today (X3)**
20.58	1	0

We first calculate for VIX = 20.58 the likelihood of SP500 going Up as

$$P(VIX = 20.58|\text{Up}) = \frac{1}{\sqrt{2\pi(20.58)^2}}\, e^{-\frac{(20.58-20.58)^2}{2(0.82)^2}} \approx 0.486$$

In a similar way for VIX = 20.58 the likelihood of SP500 going 'Down' is calculated

$$P(VIX = 20.58|\text{Down}) = \frac{1}{\sqrt{2\pi(22.56)^2}}\, e^{-\frac{(20.58-22.56)^2}{2(1.19)^2}} \approx 0.0840$$

$$\begin{aligned} P(\text{UP}|X) &= P(X_1(\text{VIX} = 20.58)|\text{Up}) \times P(X_2(\text{SP500} = 1)|\text{Up}) \\ &\quad \times P(X_3(\text{VIX} = 1)|\text{Up}) \times P(\text{Up}) \\ &\approx 0.486 \times \frac{4}{6} \times \frac{3}{6} \times \frac{1}{2} \\ &\approx 0.081 \end{aligned}$$

$$\begin{aligned} P(\text{Down}|X) &= P(X_1(\text{VIX} = 19.84)|\text{Down}) \times P(X_2(\text{SP500} = 1)|\text{Down}) \\ &\quad \times P(X_3(\text{VIX} = 1)|\text{Down}) \times P(\text{Down}) \\ &\approx 0.0000953 \times \frac{3}{4} \times \frac{2}{4} \times \frac{1}{2} \\ &\approx 0.01575 \end{aligned}$$

Thus, because the probability of $P(UP|X) > P(Down|X)$, the classifier will predict that the SP500 would open 'Up' the next day.

Gaussian Naive Bayes is particularly effective for problems with continuous data, leveraging the Gaussian distribution to estimate probabilities and make accurate predictions.

2.5 Multi-layer Perceptron

The Perceptron Algorithm is one of the earliest and most fundamental algorithms in the field of machine learning that laid the foundation for many complex and sophisticated algorithms that have been developed over the years. The perceptron algorithm is based on the concept of a single neuron in the human brain designed to mimic its process, with the input data serving as the input to the neuron and the weights representing the strength of the connections between the input neurons and the output neuron. The Perceptron is a type of linear classifier (binary classifier), which means it can be used to classify data that is linearly separable. However, the Perceptron Algorithm has some limitations, such as its inability to model complex relationships between the input features and the output class.

At its core, Multilayer Perceptron (MLP) is complex, and it has a collection of interconnected single perceptron's, also known as neurons or nodes, working together to process and analyze data.

The Multilayer Perceptron, also known as an artificial neural network, is a powerful nonlinear prediction model that can learn complex patterns in data and make accurate predictions. One of the main advantages of using a Multilayer Perceptron for stock market prediction is its ability to handle nonlinear relationships and complex patterns in the data. This allows the model to capture subtle fluctuations and trends in the stock market, which may not be easily discernible using traditional statistical models. Multi-layer perceptron's work on Feed-forward network, in which the data is passed only in one direction. Unlike, some other networks like Recurrent Neural Networks, where data is passed in both directions and forms a cycle.

The structure of an MLP can be broken down into three main parts: the input layer, the hidden layers, and the output layer.

- input layer receives the input data
- hidden layer contains a set of interconnected neurons, which process and analyze the input data passed on from the previous layer.
- output layer receives the output from the previous layers, combines them, and produces the final output.

The neurons in the input layer must be the size of the training instances, and the output layer must be the size of the output labels. However, there can be any number of neurons or layers in the hidden layer of the neural network according to the needs. The more neurons in the hidden layer the more complex problem the network can solve.

Initially when the data is fed into the network, it is first passed through the input layer and in the input layer, there is no specific operation performed but it just transfers the input into the next layer which is the hidden layer. The neurons in the hidden layer perform mathematical operations on the data and then it is passed to the next hidden layer if there is any. Finally, the processed data is passed to the output layer to produce the output.

At each of these neurons, some weights and biases are assigned. The weights are the heart of a Neural Network. These weights determine the strength of the connection between neurons. For instance, if a neuron has a high weight, it means that it has a strong connection to the next neuron and its output will have a greater impact on the final output of the network. On the other hand, if a neuron has a low weight, it means that it has a weaker

connection to the next neuron and its output will have less impact on the final output of the network.

The biases, on the other hand, are used to determine the level of activation of a neuron. It can be considered as the threshold value that a neuron must reach before it produces an output. If the input to a neuron plus its bias is greater than a certain value, the neuron will produce an output, otherwise, it will not. This allows the network to be more flexible and adaptable to different types of inputs.

However, for a Neural Network to understand the given data, it must undergo a process called learning or training, like the humans acquire knowledge. For Neural Networks to learn, an algorithm called Backpropagation is used where the error made by the network during forward propagation is sent back to the network until it reaches the input layer. During the process of propagating errors backward, the weights assigned to each of the neurons and the biases are adjusted to reduce the error, and backpropagation is used to find those weights and biases that reduce the error more often and produce an optimal result.

To understand this, we consider an example with a single input layer, one hidden layer with two activation function and an output layer.

The computation of each of the neuron with input fed inside the network can be given by:

$$z = wX + \text{Bias} \tag{2.27}$$

w = Weights of each neuron

X = inputs

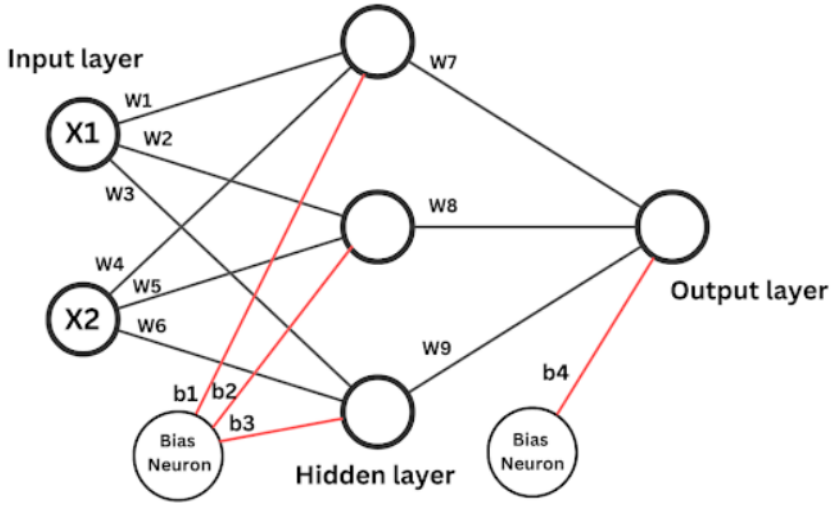

Figure 2.4: Simple MLP Structure

This is known as the weighted sum of inputs, plus a bias. So, when each neuron receives an input, it will perform the weighted sum of inputs and adds a bias and this sum of inputs plus a bias in each case is a pure linear model.

To bring non-linearity to the MLP model, a non-linear function, known as an Activation Function, is introduced. An activation function is a mathematical function that determines whether the neurons need to be activated or not by learning from really complex patterns in the given data.

Thus, in each neuron in the network, apart from finding the weighted sum of inputs and adding a bias, the results are also passed through an activation function to produce the output of each neuron which is a non-linear function. This non-linear function is then passed to the next layer and the process is repeated.

So, the output of each neuron can be represented with the activation function as below,

$$Output(y) = \varphi(wX + \text{Bias}) \tag{2.28}$$

$$z = wX + \text{Bias} \tag{2.29}$$

$$Output(y) = \varphi(z) \tag{2.30}$$

Where, φ is the activation function.

The Scikit learn Python package has the following activation function.

- **Sigmoid Function**

A mathematical function that maps any real-valued number to a value between 0 and 1, which makes it useful in binary classification problems.

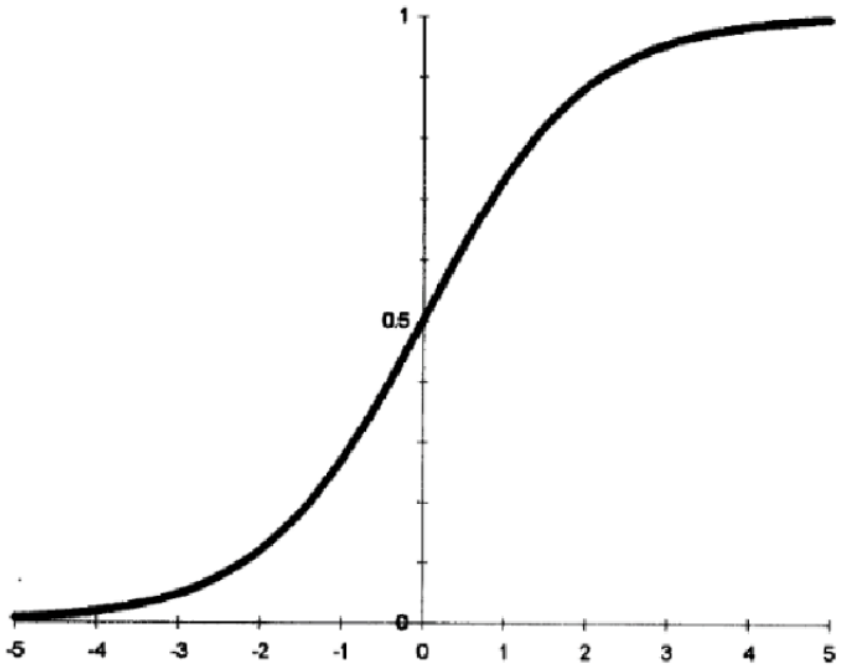

Figure 2.5: Sigmoid Activation Function

$$y = \varphi(z) = \frac{1}{1 + e^{-z}} \tag{2.31}$$

- **ReLU(Rectified Linear Unit)**

Is a simple function that returns the input value x if it is positive, and 0 otherwise. This is the default function in the Scikit learn package.

$$y = \max(0, z) \tag{2.32}$$

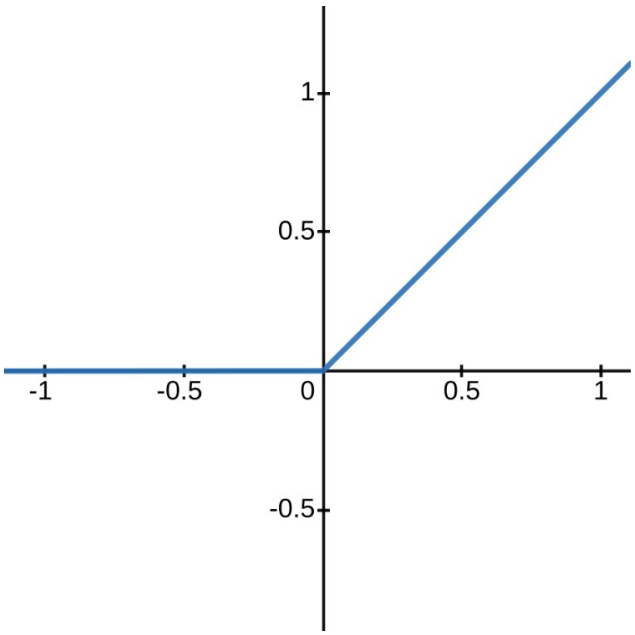

Figure 2.6: ReLU Activation Function

From the above formula, it is evident that when the value is positive it returns the value itself and if it is negative, it returns zero. Because of this simple nature, ReLU doesn't activate Neurons if it gets very small inputs which makes it a good choice for most problems.

- **Tanh (Hyperbolic Tangent)**

Similar to sigmoid but the values range from -1 to +1 centered at 0. The Tanh function is an improvement over the sigmoid function because it maps the input values to a wider range and produces a more balanced output. The mathematical formula of the Hyperbolic Tangent Function is,

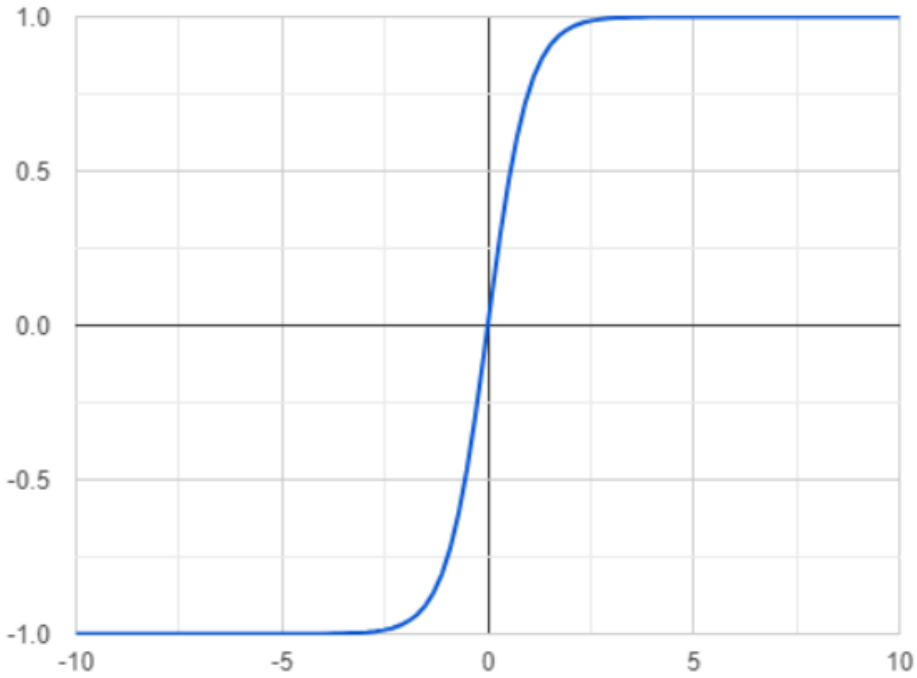

Figure 2.7: Hyperbolic Activation Function

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.33}$$

- **The Softmax Function:**

The Softmax function simply takes a vector of numbers as inputs and produces another vector containing the probability distribution of inputs.

$$y = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.34}$$

Where,

- z_i: The i-th element of the input vector z.
- $\sum_{j=1}^{n} e^{z_j}$: The sum of the exponentials of all elements in the input vector z. This acts as a normalization factor to ensure that the output probabilities sum up to 1.

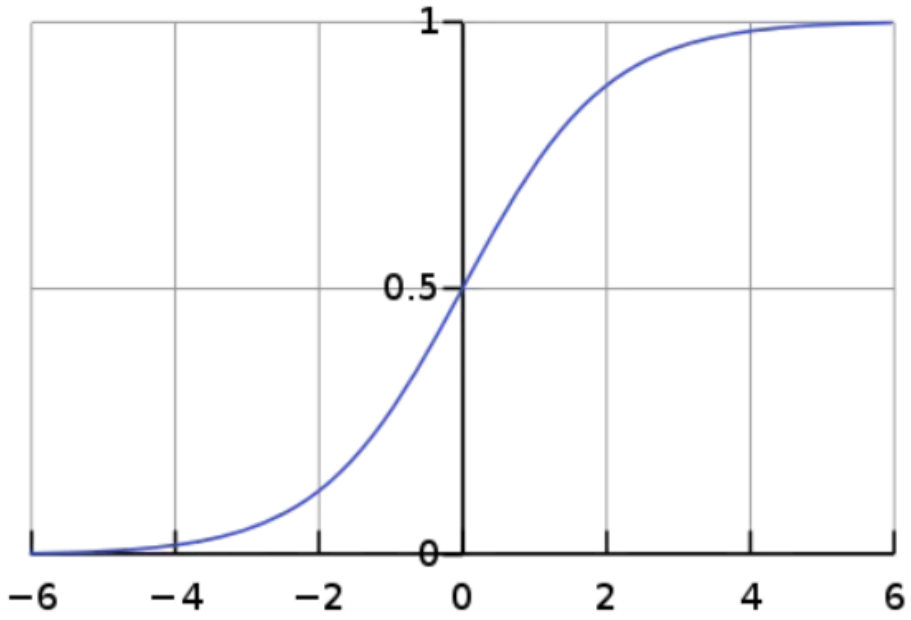

Figure 2.8: Softmax Function

Softmax function applies the exponential function to each element of the input vector, summing up, and then mapping it to the output vector. Summing the values in the output vector will equal to 1. This ensures that the output values can be interpreted as probabilities.

The output of each neuron represented with the activation function is then summed up and added with a bias and represented as below:

$$\hat{y}_t = \sum_k w_{kj} y_k + \text{Bias} \tag{2.35}$$

Where:

- $\hat{y}_t$: The predicted output of the neuron.
- $\sum_k$: The summation over all neurons in the previous layer.
- w_{kj}: The weight connecting the k-th neuron in the previous layer to the j-th neuron in the current layer.
- y_k: The output of the k-th neuron in the previous layer.
- Bias: The bias term associated with the j-th neuron in the current layer.

The next step in the process is Backpropagation. For Neural Networks to learn, an algorithm called Backpropagation is used where the error made by the network during forward propagation is sent back to the network until reaches the input layer. During the process of propagating errors backward, the weights assigned to each of the neurons and the biases are adjusted to reduce the error, and backpropagation is used to find those weights and biases that reduce the error and often produce an optimal result.

1. First the training data are split into different mini-batches(parts) and each of these training instances are fed into the input layer of the Neural Network, then the input layer sends the data into the hidden layer. The algorithm computes the output of all the Neurons in the hidden layer, and this output is passed to the next layer which is the output layer. Initially random weights from normal distribution table are selected to make those computations. Bias is assumed to be zero in this stage.

2. Next, the algorithm finds the error made by the network by comparing the outputs and the original labels. This is done through a Cost Function, which compares the predicted outputs and actual outputs(labels).

$$\text{Cost Function} = C = (y_t - \hat{y}_t) \tag{2.36}$$

Where, $\hat{y}_t$= predicted output

y_t= observed output

The errors are then summed up and squared to avoid the negative value. Additionally, for mathematical convenience, we added 1/2 . This is also called the Sum of the Squared Error Terms (SSR).

$$C = \frac{1}{2}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 \tag{2.37}$$

3. Then we compute how much the output layer neurons contributed to the error. Our main goal is to find the weights and biases that minimize the error.

To minimize the cost function, we need to understand how the cost function changes with respect to each weight (w_{kj}) and bias (Bias). This change is captured by the partial derivatives of the cost function with respect to each parameter. The partial derivative tells us the rate at which the cost changes as the parameter changes.

Applying the Chain Rule:

Chain Rule: The chain rule is a fundamental tool in calculus used to compute the derivative of a composite function. In the context of neural networks, it helps us break down the derivatives into manageable parts.

Calculating the Gradient:

Gradient for Weights: To find how much a particular weight (w_{kj}) contributed to the error, we calculate the partial derivative of the cost function (C) with respect to (w_{kj}):

$$\frac{\partial C}{\partial w_{kj}} = \frac{\partial C}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{kj}} \tag{2.38}$$

- $\frac{\partial C}{\partial \hat{y}_t}$: This term represents how the cost function changes with respect to the predicted output.
- $\frac{\partial \hat{y}_t}{\partial z_j}$: This term represents how the predicted output changes with respect to the input to the activation function (z_j).
- $\frac{\partial z_j}{\partial w_{kj}}$: This term represents how the input to the activation function changes with respect to the weight (w_{kj}).

Gradient for Biases: Similarly, for the bias, we calculate:

$$\frac{\partial C}{\partial \text{Bias}_j} = \frac{\partial C}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial z_j} \cdot \frac{\partial z_j}{\partial \text{Bias}_j} \tag{2.39}$$

- Here, $\frac{\partial z_j}{\partial \text{Bias}_j} = 1$ because the bias directly contributes to z_j.

4. Following that, we compute how much the previous layer, which is the hidden layer, contributes to the error made by the output layer, and then this process continues until it reaches the input layer. In each of these backward passes, the algorithm will find the Gradient of the error in each neuron. This is done by combining those partial derivatives derived using backpropagation from each layer into a vector known as a Gradient Vector which is a special kind of vector that points to the greatest increase of a function.

5. Finally, by using these Gradients, the weights and biases are updated accordingly through the Gradient Descent Algorithm. A Gradient Descent aim to find the steepest descent or the greatest decrease in the cost function rather than the greatest increase by taking the negative of the Gradient Vector. To find the global minimum, the algorithm will take small or large steps known as the learning rate until it reaches the optimal result. The updated weights and biases combining all the derivations is as below,

$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial C}{\partial w_{kj}} \tag{2.40}$$

$$\text{Bias}_j \leftarrow \text{Bias}_j - \eta \frac{\partial C}{\partial \text{Bias}_j} \tag{2.41}$$

Here, η is the learning rate, which controls the step size of the updates.

This process of finding the error, propagating the error backward to find the gradient, and updating the weights using the gradients is done for a specific number of iterations until we find the optimal weights and bias. In practice, MLP has multiple input nodes and hidden layer are very complex. So, when new data is fed into a Multi-Layer Perceptron (MLP), the MLP makes predictions using the derived weights and simultaneously updating its weights and biases to all its nodes to incorporate the new information. By repeating these steps with multiple new data samples, the MLP can adapt and learn from the new information, updating its internal parameters (weights) and improve on its performance.

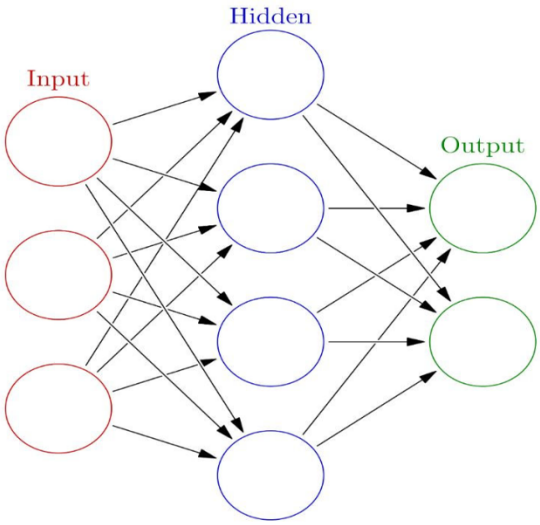

Figure 2.9: MLP Structure

2.6 Ensemble Model

Several research investigations have demonstrated that an ensemble model for time series can perform better in comparison with stand-alone 'AL' & 'ML' models. Ensemble modeling is a technique that combines predictions from multiple individual models to improve overall prediction accuracy. The rationale behind ensemble modeling is that by leveraging the collective knowledge of diverse models, the ensemble can capture different patterns and perspectives, leading to more robust and accurate predictions. In this research, we explore the application of ensemble modeling to enhance the directional prediction of the S&P 500 index by combining the predictions of individual ML models using a majority voting approach.

The main reasons for using ensemble models are:

1. Better Predictive Performance: Combining multiple models often leads to better predictive performance compared to individual models, as the ensemble can capture different patterns and perspectives that individual models may miss.

2. Reduced Overfitting: Ensemble models tend to be less prone to overfitting than individual models, as the combination of diverse models helps to average out the biases and noise present in individual models.

3. Increased Stability: Ensemble models are generally more stable and less sensitive to small changes in the training data or model parameters, making them more reliable and consistent.

There are several types of ensemble models, including:

1. Bagging (Bootstrap Aggregating): This method trains multiple models on different subsets of the training data, generated through techniques like bootstrapping or random sampling.

2. Boosting: This method sequentially trains multiple models, with each new model focusing on the instances that the previous models struggled with.

3. Stacking: This method combines the predictions of multiple models using a meta-model, which is trained on the outputs of the individual models.

4. Voting: This is a simple ensemble method where the predictions of multiple models are combined using a majority vote (for classification problems) or averaging (for regression problems).

For our predictions we have chosen the Voting method for the following reasons:

1. Ease of Implementation: Voting ensembles are relatively easy to implement and understand, as they simply require combining the predictions of individual models using a voting or averaging scheme.

2. No Additional Training: Unlike stacking or boosting, voting ensembles do not require

training an additional meta-model or sequential training, making them computationally efficient.

3. Robust Performance: Voting ensembles tend to perform well across a wide range of problems and data sets, as they leverage the diversity of the individual models without introducing complex training procedures.

4. Interpretability: The predictions of a voting ensemble can be traced back to the individual models, providing some level of interpretability and transparency.

Though ensembling across different machine learning (ML) techniques can enhance predictive performance by leveraging diverse model strengths, it can also introduces potential errors and challenges. The combined biases and variances of individual models can lead to skewed predictions, especially if data imbalances exist. The increased complexity of ensemble models can result in overfitting, while systematic errors from individual models can aggregate and propagate. Moreover, ensemble models can be difficult to interpret, complicating error diagnosis and insight generation. To mitigate these issues, strategies such as ensuring balanced training data, using diverse models, and implementing robust evaluation methods can help improve the reliability and accuracy of ensemble predictions. Despite these challenges, careful design and evaluation of ensemble models can significantly enhance their performance and robustness.

Chapter 3

Data

3.1 Overview

The research uses historical time-series data. Market data is sourced from Refinitiv Datascope. The closing prices of the selected assets (SP500, VIX, Gold, Crude Oil, 10 Year Treasuery Yield) are used to calculate various technical indicators that are used for prediction.

3.2 Market Data

The market dates and adjusted closing prices of the assets are used to calculate returns which are then used to calculate the different technical indicators of the asset. The in-sample data is used for fitting the models in order to estimate the parameters of the models, whereas the out-of-sample data is used to evaluate the forecasting performance of the models.

3.3 Other Data

VIX

The VIX, also known as the CBOE Volatility Index, is an important feature for stock prediction that is being used in our machine learning models as VIX can improve forecast accuracy. The VIX is a measure of market volatility and is often referred to as the "fear gauge" as it reflects investors' sentiment towards the market. The closing VIX futures price is considered for our model prediction.

Gold Futures, Crude Oil and 10 Year Treasuery Yield

Gold, crude oil, and interest rates are other features that are used in our prediction models as these features provide valuable insights into the macroeconomic factors that may impact stock prices. Every day closing prices of these assets are considered for prediction.

3.4 Returns

Asset returns measure the up or down trend of stocks on a day-to-day basis. There are various ways to calculate asset returns. The method of log-returns is used in this research. Historical market data from Refinitiv, specifically the adjusted closing price, is used to calculate log-returns for each asset.

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right) = \log(P_t) - \log(P_{t-1}) \tag{3.1}$$

where:

- r_t is the logarithmic return at time t.
- P_t is the closing price of the stock at time t.
- P_{t-1} is the closing price of the stock at time $t-1$.

We use log returns because they are preferred in financial analysis and they are time-additive, tend to be normally distributed, allow for symmetric relative comparisons, accurately account for compounding, and provide stability and scale independence in returns analysis.

3.5 Indicator List

Technical indicators are mathematical calculations based on the price, volume, or open interest of a security or contract. They are widely used by traders and investors to analyze price trends, generate trading signals, and make informed decisions. The use of a diverse set of technical indicators in our predictive model is based on both theoretical considerations and empirical evidence from previous studies. As Patel et al. (2015) demonstrated, the combination of multiple technical indicators can significantly enhance the accuracy of stock market predictions compared to single indicator models. This finding is further supported by Kara et al. (2011) , who showed that a model incorporating ten different technical indicators outperformed simpler models in predicting the Istanbul Stock Exchange National 100 Index. The rationale for this approach lies in the multifaceted nature of market dynamics. Different indicators capture various aspects of market behavior, from trend strength (e.g., ADX) to momentum (e.g., RSI), and from volatility (e.g., Bollinger Bands) to trend direction (e.g.,

MACD). As Nazário et al. (2017) pointed out, markets exhibit complex, non-linear behaviors that are difficult to capture with a single indicator or a simple combination of indicators. By employing a comprehensive set of indicators, our model can adapt to different market conditions and capture both short-term fluctuations and long-term trends, as evidenced by the work of Hu et al. (2015) on multiscale analysis in financial time series. In addition, the use of multiple indicators helps mitigate the risk of false signals, a common issue in technical analysis highlighted by Cervelló-Royo et al. (2015). In the context of machine learning, providing a rich feature set allows the algorithms to discover complex, non-linear relationships between indicators, potentially uncovering predictive patterns that would be invisible when using a more limited set of features. By incorporating this comprehensive set of technical indicators, this study aims to capture the full complexity of market behavior, thereby enhancing the predictive power of our model and contributing to the ongoing refinement of stock market forecasting techniques.

In this research, we derive a set of technical indicators from the closing prices of the S&P 500 index and other assets to serve as additional features for the ML models. These indicators capture various aspects of market dynamics, such as trend strength, momentum, volatility, and overbought/oversold conditions. The comprehensive list of the technical indicators used in this study, together with their descriptions, formula, and potential implications for market analysis is explained below.

Indicator	**Description**	**Formula**	**Implications**
1. Relative Strength Index (RSI)	RSI is a momentum oscillator that measures the speed and change of price movements.	$RSI = 100 - \left(\frac{100}{1 + RS}\right)$ Where $RS = \frac{\text{Average Gain}}{\text{Average Loss}}$.	$RSI > 70$ often indicates overbought conditions. $RSI < 30$ often indicates oversold conditions. Divergences between RSI and price can signal potential reversals.
2. Bollinger Bands	Bollinger Bands consist of a middle band (typically a simple moving average) with an upper and lower band set at standard deviation levels above and below the middle band.	Middle Band = 20-day SMA Upper Band = Middle Band + (2 × 20-day standard deviation) Lower Band = Middle Band (2 × 20-day standard deviation)	Price touching the upper band may indicate overbought conditions. Price touching the lower band may indicate oversold conditions. Band squeezes may precede significant price movements.

Indicator	**Description**	**Formula**	**Implications**
3. Momentum	Momentum measures the rate of change in price over a specified period.	$$\text{Momentum} = \text{Close}(t) - \text{Close}(t-n)$$ Where n is the number of periods (typically 10).	Positive momentum suggests upward trend strength. Negative momentum suggests downward trend strength. Momentum crossovers can signal potential trend changes.
4. Average Directional Movement Index (ADX)	ADX measures the strength of a trend, regardless of its direction.	$$ADX = 100 \times \frac{\text{Smoothed DX}}{\text{Smoothed TR}}$$ Where $\text{DX} = \frac{\lvert +DI - -DI \rvert}{\lvert +DI + -DI \rvert} \times 100$.	$ADX > 25$ often indicates a strong trend. $ADX < 20$ often indicates a weak trend or no trend. Rising ADX suggests increasing trend strength.

Indicator	**Description**	**Formula**	**Implications**
5. Absolute Price Oscillator (APO)	APO measures the difference between two exponential moving averages of different periods.	$APO = \text{Fast EMA} - \text{Slow EMA}$	Positive APO suggests bullish momentum. Negative APO suggests bearish momentum. APO crossovers can signal potential trend changes.
6. Aroon	Aroon measures the time since the last n-day high or low, to identify the start of new trends.	$\text{Aroon Up} = \left(\frac{N - \text{Days Since N-day High}}{N}\right) \times 100$ $\text{Aroon Down} = \left(\frac{N - \text{Days Since N-day Low}}{N}\right) \times 100$ Where N is the number of periods (typically 25).	$AroonUp > AroonDown$ suggests a bullish trend. $AroonDown > AroonUp$ suggests a bearish trend. Values above 70 indicate strong trends.

Indicator	**Description**	**Formula**	**Implications**
7. Balance of Power (BOP)	BOP measures the strength of buyers vs. sellers in the market.	$BOP = \frac{\text{Close} - \text{Open}}{\text{High} - \text{Low}}$	Positive BOP suggests buying pressure. Negative BOP suggests selling pressure. Can help confirm trend strength or potential reversals.
8. Commodity Channel Index (CCI)	CCI measures the current price level relative to an average price level over a given period.	$CCI = \frac{\text{Typical Price} - \text{SMA of Typical Price}}{0.015 \times \text{Mean Deviation}}$ Where Typical Price $= \frac{\text{High+Low+Close}}{3}$	$CCI > 100$ often indicates overbought conditions. $CCI < -100$ often indicates oversold conditions. Can help identify trend reversals or continuations.

Indicator	**Description**	**Formula**	**Implications**
9. Chande Momentum Oscillator (CMO)	CMO calculates the difference between the sum of recent gains and the sum of recent losses and expresses it as a percentage.	$CMO = 100 \times \frac{SU - SD}{SU + SD}$ Where SU = Sum of Upward Price Changes, SD = Sum of Downward Price Changes.	$CMO > 50$ may indicate overbought conditions. $CMO < -50$ may indicate oversold conditions. Can help identify potential trend reversals.
10. Directional Movement Index (DMI)	DMI helps determine if a security is trending and the strength of that trend.	$+DI = 100 \times \frac{\text{Smoothed}(+DM)}{\text{ATR}}$ $-DI = 100 \times \frac{\text{Smoothed}(-DM)}{\text{ATR}}$ Where DM = Directional Movement ATR = Average True Range	$+DI > -DI$ suggests an uptrend. $-DI > +DI$ suggests a downtrend. Crossovers can signal potential trend changes.

Indicator	Description	Formula	Implications
11. Moving Average Convergence Divergence (MACD)	MACD shows the relationship between two moving averages of a security's price.	MACD Line = 12-period EMA - 26-period EMA Signal Line = 9-period EMA of MACD Line MACD Histogram = MACD Line - Signal Line	MACD Line crossing above Signal Line is bullish. MACD Line crossing below Signal Line is bearish. Divergences can signal potential trend reversals.
12. MACD with Extended Parameters	This is a variation of the standard MACD that allows for customization of the periods used in the calculation.	Same as standard MACD, but with user-defined periods.	Similar to standard MACD, but the custom periods can be optimized for specific assets or timeframes.
13. MACD with Fixed Parameters	This is the standard MACD calculation using the default parameters.	Same as standard MACD (see 11).	Same as standard MACD (see 11).

Indicator	**Description**	**Formula**	**Implications**
14. Percentage Price Oscillator (PPO)	PPO is similar to MACD but expressed as a percentage.	$\text{PPO} = \frac{\text{12-day EMA} - \text{26-day EMA}}{\text{26-day EMA}} \times 100$	Positive PPO suggests bullish momentum. Negative PPO suggests bearish momentum. Crossovers and divergences can signal potential trend changes.
15. Rate of Change (ROC)	ROC measures the percentage change in price over a specified period.	$\text{ROC} = \frac{\text{Close} - \text{Close } n \text{ periods ago}}{\text{Close } n \text{ periods ago}} \times 100$	Positive ROC indicates upward momentum. Negative ROC indicates downward momentum. Extreme values may signal overbought or oversold conditions.

Indicator	**Description**	**Formula**	**Implications**
16. Rate of Change Percentage (ROCP)	ROCP is similar to ROC but expressed as a decimal instead of a percentage.	$\text{ROCP} = \frac{\text{Close–Close } n \text{ periods ago}}{\text{Close } n \text{ periods ago}}$	Similar to ROC, but the values are expressed as decimals.
17. Rate of Change Ratio (ROCR)	ROCR measures the ratio of the current price to the price n periods ago.	$\text{ROCR} = \frac{\text{Close}}{\text{Close } n \text{ periods ago}}$	$ROCR > 1$ indicates price increase. $ROCR < 1$ indicates price decrease. Can help identify the speed of price changes.

Indicator	**Description**	**Formula**	**Implications**
18. Stochastic Oscillator	Stochastic Oscillator compares a closing price to its price range over a given time period.	$\%K = \frac{\text{Current Close} - \text{Lowest Low}}{\text{Highest High} - \text{Lowest Low}} \times 100$ $\%D = 3\text{-day SMA of } \%K$	Readings above 80 may indicate overbought conditions. Readings below 20 may indicate oversold conditions. Crossovers between %K and %D can signal potential entry/exit points.
19. Stochastic Relative Strength Index (Stoch RSI)	Stoch RSI applies the Stochastic Oscillator formula to RSI values instead of price.	$\text{Stoch RSI} = \frac{\text{RSI} - \text{Lowest Low RSI}}{\text{Highest High RSI} - \text{Lowest Low RSI}}$	Values above 0.8 may indicate overbought conditions. Values below 0.2 may indicate oversold conditions. Can provide earlier signals than traditional RSI.

Indicator	**Description**	**Formula**	**Implications**
20. Ultimate Oscillator	Ultimate Oscillator uses multiple timeframes to avoid the pitfalls of using a single timeframe oscillator.	$\text{UO} = \frac{(4\times\text{Average}_7+2\times\text{Average}_1 4+\text{Average}_2 8)}{4+2+1}$ Where $\text{Average}_N = \frac{\text{Close}-\text{True } N \text{ Low}}{\text{True } N \text{ High}-\text{True } N \text{ Low}}$	$UO > 70$ may indicate overbought conditions. $UO < 30$ may indicate oversold conditions. Divergences can signal potential trend reversals.
21. Williams %R	Williams %R measures overbought and oversold levels, similar to the Stochastic Oscillator.	$\%\text{R} = \frac{\text{Highest High}-\text{Close}}{\text{Highest High}-\text{Lowest Low}} \times -100$	Readings from 0 to -20 suggest overbought conditions. Readings from -80 to -100 suggest oversold conditions. Can help identify potential reversals or entry/exit points.

Chapter 4

Empirical Investigation

4.1 The Quant Model & AL & ML Model

The differences between Quant, AI, and ML models can be attributed to their methodologies, data requirements, and applications. Quantitative models, also known as quant models, make use of mathematical and statistical techniques to analyze and predict financial market behavior. These models utilize equations and algorithms to code and process information extracted from datasets. On the other hand, AI and ML models focus on making predictions based on patterns found in large datasets, commonly referred to as Big Data. AI and ML models differ from quant models in their approach to learning from the data. While quant models rely on formalizing the relationships between variables for inference, AI and ML models seek to learn from the data without specific programming. They utilize machine learning algorithms that search for patterns within the data and make accurate predictions based on those patterns. AI and ML models also differ from quant models in terms of the

complexity of their behavior. Whereas quant models are primarily concerned with financial market behavior and predicting outcomes in a specific domain, AI models aim to create intelligent models that can simulate human cognition and perform actions to achieve a specific objective. Additionally, ML models are a subfield of AI that focus on learning and improving predictions based on passive observations. In summary, quant models rely on mathematical and statistical techniques to analyze financial market behavior and make predictions, while AI and ML models use machine learning algorithms to identify patterns in data and make predictions based on those patterns.

4.2 Experimental Setup

Python was extensively used for our experiments. Scikit learn library has different packages that can be called to perform our classification tasks. The following is a brief overview of the experiment setup.

1. Market closing price data are used for the different asset class i.e SP500, Gold, VIX Oil and 10 Years US treasury. The period of study was for 5 years from 01-01-2017 to 31-12-2022 during which the predictions are made.

2. From the above asset class different technical indicators were established which are used as additional features to make prediction. The TA-Lib python package was used to derive these features.

3. The models are trained on a rolling window basis of 250 days. For training, data from 01-01-2016 to 31-12-2017 are used. Once they are trained and fit, they are applied to

the testing dataset on a rolling basis to make daily predictions.

4. The predictions are then evaluated and analysed using the Confusion Matrix and Accuracy score for the different models.

5. In the final step we Ensemble all the models to check if there is improvement in prediction accuracy.

4.3 Instantiating Results

The Confusion matrix and accuracy scores are the main parameters against which all the 5 models were analysed. A confusion matrix is a table that summarizes the performance of a classification model by showing the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions and is commonly used to evaluate predictions of machine learning model.

The Confusion matrix is constructed as below where,

True Positive (TP) is the number of instances that are actually positive and are correctly predicted as positive by the model.

True Negative (TN) is the number of instances that are actually negative and are correctly predicted as negative by the model.

False Positive (FP) is the number of instances that are actually negative but are incor-

rectly predicted as positive by the model. Also known as a Type I error.

False Negative (FN) is the number of instances that are actually positive but are incorrectly predicted as negative by the model. Also known as a Type II error.

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Table 4.1: Confusion Matrix

Similarly, the Accuracy score is another metric used to measure the performance of a classification model. It calculates the proportion of correct predictions made by the model out of the total number of predictions.

The formula to calculate the accuracy score is:

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

Accuracy score is typically represented as a value between 0 and 1, where a score of 1 indicates perfect accuracy.

Individual Model Performance:

Initially the experiment is run on individual models and predictions are made. Below the performance of individual models are summarised,

Table 4.2: Individual Model Performance

Model	Accuracy	Confusion Matrix
Gaussian NB	53.42%	$\begin{bmatrix} 44 & 643 \\ 59 & 761 \end{bmatrix}$
Random Forest	50.90%	$\begin{bmatrix} 316 & 371 \\ 369 & 451 \end{bmatrix}$
Decision Tree	53.02%	$\begin{bmatrix} 366 & 321 \\ 387 & 433 \end{bmatrix}$
MLP	52.16%	$\begin{bmatrix} 320 & 367 \\ 354 & 466 \end{bmatrix}$
Logistic Regression	51.69%	$\begin{bmatrix} 181 & 506 \\ 222 & 598 \end{bmatrix}$

- The individual model accuracies range from 50.9% (Random Forest) to 53.42% (Gaussian NB), which are relatively low for stock market prediction tasks.
- Gaussian Naive Bayes has the highest accuracy score of 53.42(%), however by analysing the confusion matrix it noted that the matrix is highly skewed to Down prediction. The confusion matrix for the other models is evenly distributed.

- Decision Tree has the highest accuracy score of 53.02(%) if we consider the predictions in conjecture with the confusion matrix.
- Random Forest has the lowest accuracy score of 50.90(%).
- Logistic Regression and MLP models perform better than the Random Forest.

The results presented for each model were obtained using the default settings provided by the Python scikit-learn package. The summary of the default parameters used in different models is presented in the table below.

These default configurations serve as a baseline for our experiments and provide a starting point for understanding the performance of each model. It's important to note that while these default settings are generally designed to work reasonably well across a wide range of problems, they may not be optimal for our specific task of predicting S&P 500 index movements.

Model	Hyperparameter	Default Value	Impact/Options
Gaussian Naive Bayes	var_smoothing	1e-9	Adds stability by using a small portion of the largest variance.
Random Forest	Number of estimators	100	Number of trees in the forest.
	Max depth	None	Controls the maximum depth of each tree.
	Min samples split	2	Minimum samples required to split an internal node.
	Min samples leaf	1	Minimum samples required at a leaf node.
	Max features	'auto'	Number of features considered for the best split.
Decision Tree	Criterion	'gini'	Options: 'gini', 'entropy' - Function to measure the quality of a split.
	Max depth	None	Maximum depth of the tree.
	Min samples split	2	Minimum samples required to split an internal node.
	Min samples leaf	1	Minimum samples required at a leaf node.
	Max features	None	Number of features considered for the best split.

Continued on next page

Model	Hyperparameter	Default Value	Impact/Options
Multi-Layer Perceptron	Number of units in hidden layer	100	Determines model's capacity to learn complex patterns.
	Activation function	ReLU	Options: ReLU, tanh, logistic - Introduces non-linearity for complex relationships.
	L2 regularization parameter (alpha)	0.0001	Helps prevent overfitting by adding a penalty term.
	Initial learning rate	0.001	Influences how quickly the model learns.
	Solver	Adam	Options: LBFGS, SGD, Adam - Optimization algorithm for finding optimal weights.
Logistic Regression	Penalty	'l2'	Options: 'l1', 'l2', 'elasticnet', 'none' - Specifies the norm used for penalization.
	C (inverse of regularization strength)	1.0	Smaller values specify stronger regularization.
	Solver	'lbfgs'	Options: 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' - Algorithm for optimization.
	Max iterations	100	Maximum iterations for solver convergence.

Continued on next page

Model	**Hyperparameter**	**Default Value**	**Impact/Options**
	Multi class	'auto'	Options: 'auto', 'ovr', 'multinomial' - Specifies how multi-class classification is handled.

Hyperparameter Tuning of MLP Model:

In machine learning, hyperparameter tuning is a critical step in optimizing model performance. Hyperparameters are configuration variables that are external to the model and whose values cannot be estimated from the data. They are typically set before the learning process begins and significantly influence the model's learning process and, consequently, its performance. The importance of hyperparameter tuning lies in its ability to adapt a general-purpose model architecture to a specific problem domain. In complex tasks such as financial market prediction, where the relationships between variables are often non-linear and time-dependent, the default hyperparameters of a model may not be optimal. Tuning these hyperparameters allows us to find the best configuration that balances the model's ability to capture intricate patterns in the data with its capacity to generalize to unseen data. This process can lead to substantial improvements in predictive accuracy and model robustness, which are crucial in the high-stakes environment of financial forecasting.

In our initial study, we evaluated various machine learning models, including a Multi-Layer Perceptron (MLP), for predicting the directional movement of the S&P 500 index using default parameter setting. The MLP model with default parameters achieved an accuracy of 52.16%.

The primary motivations for conducting hyperparameter tuning were:

1. To enhance the model's predictive accuracy
2. To adapt the model to the specific characteristics of financial time series data
3. To find the optimal balance between model complexity and generalization

4. To gain insights into which model parameters are most crucial for this specific prediction task

By systematically exploring different hyperparameter configurations, we aim to not only improve the model's performance but also to deepen our understanding of how different aspects of the model architecture interact with the unique challenges posed by financial market prediction.

Methodology:

Our methodology combined several techniques to efficiently tune hyperparameters while maintaining the integrity of our time series prediction task. Hyperparameter tuning is computationally intensive, often requiring multiple model trainings across various parameter combinations, which can be prohibitively time-consuming for large datasets. To manage this complexity, we implemented feature selection to focus on the most relevant predictors, reducing the dimensionality of the input space. Additionally, we employed a reduced rolling window approach, which allowed us to capture temporal patterns while significantly decreasing computation time. These strategies enabled a more thorough exploration of the hyperparameter space within reasonable time constraints, balancing the trade-off between model optimization and computational feasibility.

1. Optimization Approach :

We employed the Optuna library for hyperparameter optimization. Optuna uses a Bayesian optimization approach to efficiently search the hyperparameter space. We configured Optuna to run 100 trials, each testing a different combination of hyperparameters.

2. Feature Selection :

Prior to hyperparameter tuning, we performed feature selection using SelectKBest python package. This process reduced our feature set to the top 20 most relevant features. The goals of this step were to:

- Improve computational efficiency

- Reduce noise in the input data

- Focus the model on the most informative features

3. Reduced Rolling Window Method :

To evaluate the model's performance across various time periods while managing computational complexity, we implemented a reduced rolling window approach:

- A smaller window size of 125 days was used in the experiment.

Our model training and testing followed a rolling-window approach. With the reduced window size, we train the model and then test its performance on the next day's market movement. This window was then rolled forward one day, the model retrained, and the process was repeated throughout the dataset, similar to the original experiment setup.

Results and Analysis

A total of 100 trials were performed with Optuna. The following is the performance distribution of important parameters.

Performance Distribution

1. Activation Function Performance:

    - Logistic: Average accuracy of 52.8% (across 52 trials)

    - Tanh: Average accuracy of 50.1% (across 24 trials)

- ReLU: Average accuracy of 53.2% (across 24 trials)

2. Solver Performance:

   - LBFGS: Average accuracy of 51.7% (across 37 trials)
   - SGD: Average accuracy of 52.2% (across 40 trials)
   - Adam: Average accuracy of 52.7% (across 23 trials)

Top Performing Configurations

1. Logistic activation, LBFGS solver: 55.21% accuracy (Trial 26)
2. Logistic activation, LBFGS solver: 55.18% accuracy (Trial 27)
3. Logistic activation, LBFGS solver: 55.16% accuracy (Trial 34)

The hyperparameter tuning process for the MLP model in S&P 500 prediction revealed that the logistic activation function consistently outperformed ReLU and tanh, while the SGD and LBFGS solvers showed strong performance. The best configuration (logistic activation with the LBFGS solver) achieved a accuracy of 55.21%, representing a 3.05 percentage point improvement over the original model with default parameters. This modest but significant improvement underscores the importance of hyperparameter optimization in financial prediction tasks.

Furthermore, the number of hidden units in the best-performing models ranged from 50 to 98, indicating that relatively small to medium-sized hidden layers were sufficient for this

prediction task. This could be attributed to the nature of the financial time series data, where overly complex models might lead to overfitting.

Additionally, the learning rates for the top configurations were consistently low (around 0.0005-0.0006), suggesting that small, careful steps in the optimization process led to better performance. This is typical in financial prediction tasks, where the signal-to-noise ratio can be low, and aggressive learning rates might cause the model to overshoot optimal solutions.

Thus, these findings highlight the nuanced nature of hyperparameter tuning in financial prediction tasks. While general guidelines can be helpful, the optimal configuration often depends on the specific characteristics of the data and the prediction task at hand. This underscores the value of systematic hyperparameter tuning in developing effective machine learning models for financial forecasting.

For future work, similar hyperparameter tuning could be applied to the other models, potentially leading to improved performance across the board. The ranges and options provided here could serve as a starting point for such optimization efforts.

Validation of Tuned Model on Original Dataset

After identifying the best hyperparameters through the reduced rolling-window approach, we applied these optimal settings (logistic activation function and LBFGS solver) to the MLP model and ran it on the original, full dataset using the actual rolling-window setup. This process allowed us to validate the effectiveness of the tuned hyperparameters on the entire data set, ensuring that the improvements observed during the tuning process translated to enhanced performance on the entire time series.

Top Configuration (Logistic activation, LBFGS solver)

- **Accuracy:** 55.28%
- **Confusion Matrix:**

$$\text{Confusion Matrix} = \begin{bmatrix} 77 & 610 \\ 64 & 756 \end{bmatrix}$$

Default Configuration (ReLU activation, Adam solver)

- **Accuracy:** 52.16%
- **Confusion Matrix:**

$$\text{Confusion Matrix} = \begin{bmatrix} 320 & 367 \\ 354 & 466 \end{bmatrix}$$

The top configuration (Logistic activation, LBFGS solver) achieved a 3.12 percentage point higher accuracy than the default configuration (ReLU activation, Adam solver). However, the confusion matrices reveal significant differences in prediction patterns. The top configuration shows a strong bias towards predicting upward movements, with higher true positives (756 vs 466) and lower false negatives (64 vs 354), but also higher false positives (610 vs 367) and lower true negatives (77 vs 320). This suggests that while the top configuration is more sensitive to upward movements, it may be less reliable in identifying downward trends, potentially making it more suitable for bullish market conditions. Additionally, the natural tendency of the logistic activation function to push outputs towards binary extremes may contribute to this polarized prediction pattern.

Ensemble:

To improve the prediction performance, an ensemble modeling approach was adopted, where majority voting mechanism is used to combine predictions from five distinct machine learning models: Gaussian Naive Bayes, Random Forest, Decision Tree, Multi-Layer Perceptron (MLP), and Logistic Regression. This approach is designed to harness the unique strengths of each individual model while potentially offseting their respective weaknesses.

Key aspects of this ensemble approach include:

- Diversity of models: The chosen models represent a range of algorithmic approaches, from probabilistic (Naive Bayes) to tree-based (Random Forest, Decision Tree) to neural network (MLP) and linear models (Logistic Regression).

- Majority voting: This simple yet effective method allows for a democratic decision-making process among the models.

- Multiple voting scenarios: By considering different levels of agreement among the models, the approach provides flexibility in how predictions are combined.

The heart of the ensemble method lies in its voting mechanism, which combines the individual model predictions into a single, potentially more robust, prediction. This process is executed for each prediction day throughout the study period. The voting mechanism operates as follows:

Individual model predictions:

On each prediction day, all five models independently generate their predictions (UP or DOWN) based on their respective algorithms and the training data from the rolling window.

Combination of predictions:

These individual predictions are then aggregated using several voting scenarios, each representing a different level of agreement among the models.

Voting scenarios:

1. **Unanimous agreement:** All 5 models predict the same direction.

   This scenario represents the highest level of consensus among the models. While potentially the most reliable, it may occur relatively infrequently.

2. **Strong majority:** 4 out of 5 models agree on the prediction.

   This scenario balances strong consensus with increased frequency of occurrence.

3. **Simple majority:** 3 out of 5 models agree on the prediction.

   This represents the minimum level of agreement needed for a majority decision. It ensures a prediction is made for every day, even when there's significant disagreement among models.

4. **Top model consensus:** All top 3 performing models agree.

   This scenario prioritizes the models that have shown the best individual performance. The top 3 models are determined based on their accuracy over the entire study period.

5. **Top model majority:** 2 out of the top 3 performing models agree.

   This scenario balances the emphasis on top-performing models with increased prediction frequency.

Final prediction determination:

For each day, the final prediction is recorded for each of these voting scenarios. This allows analysis of how different levels of model agreement correlate with prediction accuracy.

By implementing this detailed ensemble voting mechanism, the study aims to extract maximum value from the combination of diverse machine learning models. This approach not only potentially improves the accuracy of the overall prediction, but also provides insights into the relative performance of different levels of model consensus in predicting stock market movements.

The results from different Ensemble scenario are summarized below :

Table 4.4: Ensemble Modeling Performance

Ensemble Scenario	Predictions Made	Overall Accuracy	Confusion Matrix
All 5 models agree	236	58.90%	$\begin{bmatrix} 5 & 90 \\ 7 & 134 \end{bmatrix}$
4 of 5 models agree	754	54.91%	$\begin{bmatrix} 43 & 291 \\ 49 & 371 \end{bmatrix}$
3 of 5 models agree	1507	53.48%	$\begin{bmatrix} 186 & 501 \\ 200 & 620 \end{bmatrix}$
Top 3 models.	436	57.56%	$\begin{bmatrix} 15 & 171 \\ 14 & 236 \end{bmatrix}$
2 of Top 3 models agree	1507	54.21%	$\begin{bmatrix} 199 & 488 \\ 202 & 618 \end{bmatrix}$

The ensemble model's performance varied across different agreement scenarios. When all five models agreed (236 out of 1507 instances), it achieved the highest accuracy of 58.90%, correctly predicting 134 upward movements and 5 downward movements. As agreement levels decreased, accuracy generally declined. With four models agreeing (754 instances), accuracy dropped to 54.91% (371 correct upward, 43 correct downward predictions). When three models agreed (all 1507 instances), accuracy further decreased to 53.48% (620 correct upward, 186 correct downward predictions). Using only the top three models, unanimous agreement (436 instances) yielded 57.56% accuracy (236 correct upward, 15 correct downward predictions), while two out of three agreeing (all 1507 instances) resulted in 54.21% accuracy (618 correct upward, 199 correct downward predictions).

Across all ensemble scenarios, a consistent pattern emerged: the upward market prediction accuracy was higher than the downward market prediction accuracy. This observation suggests that the ensemble models were better at predicting upward market movements compared to downward movements.

Several other factors could contribute to this discrepancy, such as:

1. Asymmetric patterns in the data: The stock market may exhibit different patterns or dynamics during upward and downward trends, making it easier for the models to capture upward movements compared to downward movements.
2. Imbalanced data: If the dataset had a higher proportion of instances representing upward market movements, the models may have been biased towards predicting upward movements

more accurately.

3. Model biases: The individual models themselves have inherent biases or assumptions that make them more effective in predicting upward market movements compared to downward movements. From careful examination of the the up and down accuracy scores, we observed that the Gaussian Naive Bayes followed by Logistic Regression exhibits highly skewed distribution, potentially introducing biases in the predictions.

Table 4.5: Model Accuracy Summary

Models / Scenario	Overall Accuracy	Down Accuracy	Up Accuracy
Gaussian NB	53.42%	6.41%	92.80%
Random Forest	50.90%	45.98%	54.94%
Decision Tree	53.02%	53.27%	52.80%
MLP	52.16%	46.52%	56.83%
Logistic Regression	51.69%	26.35%	72.93%
All 5 models agree	58.90%	5.26%	95.04%
4 of 5 models agree	54.91%	12.87%	88.33%
3 of 5 models agree	53.48%	27.07%	75.61%
Top 3 models	57.56%	8.06%	94.40%
2 of Top 3 models agree	54.21%	28.97%	75.37%

4. Market dynamics: The underlying factors influencing upward and downward market movements may be different, and the models may have captured the patterns associated

with upward movements more effectively.

It's important to note that the varying number of predictions made across different ensemble scenarios could impact the reliability and generalisation of the results. Scenarios with a lower number of predictions may not provide a comprehensive representation of the models' performance.

Furthermore, it is also important to note that the results reported for the models were obtained using a standard configuration without extensive hyperparameter tuning. Due to the rolling window approach used in our study, traditional methods of monitoring loss convergence and evaluating performance on test sample set were not directly applicable. The dynamic nature of financial markets and our prediction methodology make it challenging to interpret the model's learning process in the same way as static datasets.

The use of a rolling window approach, while beneficial for capturing evolving market dynamics, introduces complexities in assessing model convergence and stability. Each prediction is made using a model trained on the most recent historical data, which means the training set is constantly changing. This makes it difficult to perform typical diagnostics like learning curve analysis or validation on a fixed test set.

Furthermore, we acknowledge that the influence of architectural parameters (such as the number of layers, neurons per layer, and activation functions) on the models performance was not extensively explored in this study. The results presented reflect the performance of a baseline configuration, and there may be potential for improvement.

Chapter 5

Conclusion and Discussion

The conducted research aimed to improve stock market direction prediction accuracy by employing an ensemble modeling approach. Five individual models (Gaussian Naive Bayes, Random Forest, Decision Tree, Artificial Neural Network, and Logistic Regression) were initially evaluated, and their individual performances were found to be relatively low, with accuracy's ranging from 50.9% to 53.42%.

Since all the model results we based on default parameters, hyperparameter tuning was conducted on the Multi-Layer Perceptron (MLP) model to optimize its performance and demonstrate the potential for improvement through careful model configuration. The tuning focused on key parameters such as the number of hidden layer units, the activation function, the regularization strength, the learning rate, and solver algorithm. Using the Optuna library for an efficient hyperparameter search, 100 trials were performed on a reduced dataset. The best configuration, using the logistic activation function and the LBFGS solver, achieved an accuracy of 55.28% when applied to the entire dataset, representing a significant

improvement of 3.12 percentage points over the default configuration (52.16%). This substantial enhancement underscores the importance of model optimization and suggests that similar tuning of other models could potentially yield further improvements in prediction accuracy.

To enhance the prediction performance, an ensemble modelling technique was implemented, where the final prediction was based on the majority voting of the individual models. Different ensemble combinations were explored, ranging from scenarios where all models agreed on the market direction to scenarios where only two out of the top three models agreed.

The results demonstrated that the ensemble modelling approach generally improved the prediction accuracy compared to individual models. The highest overall accuracy of 58.89% was achieved when all five models agreed on the market direction. However, it is important to note that the number of predictions made in this scenario was relatively low, which could limit the generalisation of the results.

A consistent trend observed across all ensemble scenarios was that the upward market prediction accuracy was higher than the downward market prediction accuracy. This discrepancy could be attributed to various factors, such as asymmetric patterns in the data, imbalanced datasets, inherent model biases, or differences in the underlying factors influencing upward and downward market movements.

While the ensemble modelling approach showed potential for improving stock market direction prediction accuracy, the achieved accuracy levels were still relatively moderate, highlighting the complexity of stock market prediction tasks and the need for further im-

provements and calibrations.

Future Research:

1. Investigate the underlying causes of the discrepancy between upward and downward market prediction accuracies:

- Analyze the data for potential imbalances or asymmetric patterns. - Explore techniques such as data augmentation or oversampling to address imbalanced datasets.

- Investigate the individual model biases and assumptions that may contribute to the observed discrepancy.

2. Explore advanced ensemble modelling techniques:

- Investigate ensemble methods that assign different weights to individual models based on their performance or confidence levels.

- Explore stacking techniques, where the outputs of individual models are used as input features for a meta-model.

- Implement dynamic ensemble selection methods, where the ensemble composition is adaptively adjusted based on the input data characteristics.

3. Incorporate domain-specific knowledge and alternative data sources:

- Leverage domain expertise and financial theory to engineer relevant features or constraints for the models.

- Explore the integration of alternative data sources, such as news, social media, or sentiment analysis, to capture market sentiment and events.

4. Explore advanced machine learning techniques:

- Investigate deep learning architectures, such as convolutional neural networks or recur-

rent neural networks, which may capture complex patterns in financial time series data.

- Explore transfer learning techniques, where models pre-trained on large datasets can be fine-tuned for stock market prediction tasks.

- Investigate the application of reinforcement learning techniques for dynamic portfolio management and trading strategies.

5. Conduct rigorous evaluation and model validation:

- Implement robust model evaluation techniques, such as cross-validation or out-of-sample testing, to assess the generalization performance of the models.

- Investigate the impact of different evaluation metrics and cost functions tailored to the specific requirements of stock market prediction tasks.

By addressing the limitations of the current research and exploring advanced techniques, future studies can potentially improve the accuracy and reliability of stock market direction prediction models, contributing to more informed investment decisions and portfolio management strategies.

Furthermore, this research can be used as a foundation for developing stock trading strategies by incorporating the ensemble modeling approach and leveraging the predictions made by the different models. Here are some potential ways to integrate this research into trading strategies:

1. Directional trading: The ensemble model predictions can be used to determine the directional bias for the stock market, allowing traders to take positions accordingly. For instance, if the ensemble model predicts an upward market movement, traders could consider taking long positions or buying call options.

2. Risk management: The ensemble model predictions can be used as an additional risk management tool. Traders could adjust their position sizes or implement stop-loss levels based on the predicted market direction, aiming to minimize potential losses during unfavorable market conditions.

3. Portfolio optimization: The ensemble model predictions could be incorporated into portfolio optimization strategies, adjusting the allocation of assets based on the predicted market direction. For example, during predicted upward market movements, the portfolio could be rebalanced to increase exposure to equities, while during predicted downward movements, the portfolio could be shifted towards defensive assets or cash.

4. Trading strategies: The ensemble model predictions could be combined with other technical indicators or fundamental analysis to develop more comprehensive trading strategies. For example, traders could use the ensemble model predictions as a filter, entering trades only when the predicted market direction aligns with other technical or fundamental signals. It is important to note that while the ensemble modeling approach shows promise, it should not be solely relied upon for trading decisions. Traders should always consider other factors, such as risk management, diversification, and their overall investment objectives, when developing and implementing trading strategies.

Bibliography

Altman, N. and Krzywinski, M. (2017). Ensemble methods: bagging and random forests. *Nature Methods*, 14(10):933–935.

Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th international conference on computer modelling and simulation*, pages 106–112. IEEE.

Ballings, M., Van den Poel, D., Hespeels, N., and Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert systems with Applications*, 42(20):7046–7056.

Bramer, M. (2007). Avoiding overfitting of decision trees. *Principles of data mining*, pages 119–134.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Campbell, J. Y. (1987). Stock returns and the term structure. *Journal of financial economics*, 18(2):373–399.

Cervelló-Royo, R., Guijarro, F., and Michniuk, K. (2015). Stock market trading rule based

on pattern recognition and technical analysis: Forecasting the djia index with intraday data. *Expert systems with Applications*, 42(14):5963–5975.

Chong, E., Han, C., and Park, F. C. (2017). Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187–205.

Deng, S., Mitsubuchi, T., Shioda, K., Shimada, T., and Sakurai, A. (2011). Combining technical analysis with sentiment analysis for stock price prediction. In *2011 IEEE ninth international conference on dependable, autonomic and secure computing*, pages 800–807. IEEE.

Dutta, A., Bandopadhyay, G., and Sengupta, S. (2012). Prediction of stock performance in the indian stock market using logistic regression. *International Journal of Business and Information*, 7(1):105.

Fiévet, L. and Sornette, D. (2018). *Decision trees unearth return sign predictability in the S&P 500*, volume 18. Taylor & Francis.

Gu, S., Kelly, B., and Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.

Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., and Soman, K. (2018). Nse stock market prediction using deep-learning models. *Procedia computer science*, 132:1351–1362.

Hosmer, D. W. (2000). *Applied logistic regression*. John Wiley & Sons.

Hu, Y., Feng, B., Zhang, X., Ngai, E., and Liu, M. (2015). Stock trading rule discovery with an evolutionary trend following model. *Expert Systems with Applications*, 42(1):212–222.

Kara, Y., Boyacioglu, M. A., and Baykan, Ö. K. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert systems with Applications*, 38(5):5311–5319.

Khedr, A. E., Yaseen, N., et al. (2017). Predicting stock market behavior using data mining technique and news sentiment analysis. *International Journal of Intelligent Systems and Applications*, 9(7):22.

Kilian, L. and Park, C. (2009). *The impact of oil price shocks on the US stock market*, volume 50. Wiley Online Library.

Leung, M. T., Daouk, H., and Chen, A.-S. (2000). Forecasting stock indices: a comparison of classification and level estimation models. *International Journal of forecasting*, 16(2):173–190.

Mensi, W., Beljid, M., Boubaker, A., and Managi, S. (2013). Correlations and volatility spillovers across commodity and stock markets: Linking energies, food, and gold. *Economic Modelling*, 32:15–22.

Mitra, L. and Mitra, G. (2011). *Applications of news analytics in finance: A review*. Wiley Online Library.

Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.

Nazário, R. T. F., e Silva, J. L., Sobreiro, V. A., and Kimura, H. (2017). A literature review of technical analysis on stock markets. *The Quarterly Review of Economics and Finance*, 66:115–126.

Patel, J., Shah, S., Thakkar, P., and Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert systems with applications*, 42(1):259–268.

Patel, P. B. and Marwala, T. (2006). Forecasting closing price indices using neural networks. In *2006 IEEE international conference on systems, man and cybernetics*, volume 3, pages 2351–2356. IEEE.

Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46.

Sarantis, N. (2001). *Nonlinearities, cyclical behaviour and predictability in stock markets: international evidence*, volume 17. Elsevier.

Shynkevich, Y., McGinnity, T. M., Coleman, S. A., Belatreche, A., and Li, Y. (2017). Forecasting price movements using technical indicators: Investigating the impact of varying input window length. *Neurocomputing*, 264:71–88.

Smales, L. A. (2017). *The importance of fear: investor sentiment and stock market returns*, volume 49. Taylor & Francis.

Strobl, C., Boulesteix, A.-L., and Augustin, T. (2007). Unbiased split selection for classifica-

tion trees based on the gini index. *Computational Statistics & Data Analysis*, 52(1):483–501.

Wang, H. (2019). Vix and volatility forecasting: A new insight. *Physica A: Statistical Mechanics and its Applications*, 533:121951.