University of Essex

Department of Mathematical Sciences

MA981: Dissertation

Forecasting the S&P 500 Index: A Comparison of Long Short-Term Neural Networks, Support Vector Machines, Random Forests, and Feed Forward Neural Networks

Registration number: 1906970

Supervisor: Dr. John O'Hara

Word count: 10131

Contents

Abstract

Industrial and academic finance, alike, have become inextricably linked to the application of machine learning models to the study of financial time-series. By extension, investors and academics, have sought to develop the best model to gain the greatest advantage over the market, if but for a moment. Nevertheless, and despite being driven by the 20th and 21st Century innovations in computer engineering and computer science, such a model has not been produced. Instead, the financial literature is rife with papers, examining the application of evergreen and novel models to specific scenario, with little room for cross-study comparison of results, except in the case of a handful of open-source papers available on Git-hub. In this paper, we, i), survey the sparse developments in the literature on financial forecasting, ii), attempt to derive conclusions from it, and, iii) implement three of the most commonly deployed models to forecast the daily directional movements of the S&P 500 Index (GSPC). Namely, we apply a Random Forest (RF) with XGBoost, a Long Short-Term Memory Neural Network (LSTM), and Support Vector Classifier (SVC) with rbf-kernel, compare them according to four metrics, and see how they fair as compared to the bench-mark of a Feed Forward Neural Network (FFNN).

Keywords: Support Vector Machines, LSTM, Random Forest, Stock Price Prediction, S&P 500 Index.

1 Introduction

Developments in optimization algorithms, model architectures, and computational power, have reformed the field of quantitative finance, allowing modellers to exploit more sophisticated methods for financial forecasting [1]. The first step in the direction of such computational models came in 1973, when finance was given a quantitative foundation on which to stand by Black, Merton, and Scholes, who, based on Kyoshi Ito's work on differentiation with respect to a stochastic variable, produced their namesake model for option pricing [2]. At that time, the Cold War's technologically competitive landscape had driven more capital and resources into the physical sciences, yielding advances in electronics and computing, such as integrated circuits with more transistors. It was not until the mid-1990s, however, that the fall of the Soviet Union and the receding waves of its technological zeitgeist brought such computational power together with innovative modelling methods being developed in finance [2]. With the end of the war, physicists, astronomers, and mathematicians, thitherto focused on applying stochastic differential equations to statistical mechanics and astrophysics, became intrigued by the complex dynamics of markets – and the financial interests to-which those markets were married.

Since then, the influx of investment and pecuniary opportunities to the financial industry has spurred advancements within the academic instantiations of the field, prompting a shift of focus within economics departments to quantitative modelling and econometric methods. Though the adoption of these methods within departments was rapid, development of appropriate best-practices to accompany them followed a relatively more gradual trajectory; by 2003, the inappropriateness of classical parametric model assumptions to non-stationary, noisy, and lepto-kurtic time-series data had cast doubts on several research applications of regression modelling across the macroeconomic literature [3, 4]. This is especially true inasmuch as such research concerns financial time-series, which are additionally characterized by non-linearity and complex dependencies [5]. Those methods have been superannuated by more sophisticated and appropriate statistical models, such as the Auto-regressive Integrated Moving Average (ARIMA) and the Exponential Smoothing models (ES), which better account for non-stationary, trending, and seasonal properties of data [6].

Furthermore, alternatives have emerged from the field of computer science over the past 20 years, as improvements in computer microchip efficiency, in accordance with Moore's Law, produced a revolution in computational intelligence. The result has been a surge in the volume and variety of successful applications of both machine and deep learning to computer vision, natural language processing, and, relevantly, quantitative finance [5].

More recently, the general consensus borne out by the financial forecasting literature is that machine learning models, such as Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Random Forest (RF) outperform traditional statistical models on financial forecasting tasks [7, 5, 6]. It is the purpose of this research to (i) provide an overview and definition of machine learning, deep learning, and statistical models that have shown the most promise for financial forecasting, (ii) survey developments in each of their respective applications to financial forecasting, evaluating the consensus borne out by the literature, and (iii) use the performance of a regular FFNN model as a benchmark for a performance comparison between a Support Vector Machine with Radial Basis Function (rbf) Kernel, a Long-Short Term Memory Neural Network (LSTM), and a Random Forest model with XGBoost on the task of forecasting the returns of the GSPC Index.

2 Overview of Machine Learning Models to be Compared

2.1 Support Vector Machines

The Support Vector Machine (SVM) was developed in 1995 [8], as a highly versatile supervised learning model, best fitted to data with high-dimensional feature vectors [9]. It converts classification questions into quadratic convex programming problems, which means their solutions are guaranteed to be global optima [10]. This provides SVMs a significant advantage over other machine learning algorithms, such as Artificial Neural Networks, which employ myopic algorithms, such as Gradient Descent and are, therefore, likely to converge upon a local optimum. Furthermore, by mapping data in higher dimensions and applying transformations to them using non-linear functions, SVMs can accommodate a wide range of data structures and sizes [11]. Once mapped, the data are fitted with a separating hyper-plane, an n-1 dimensional manifold embedded within the

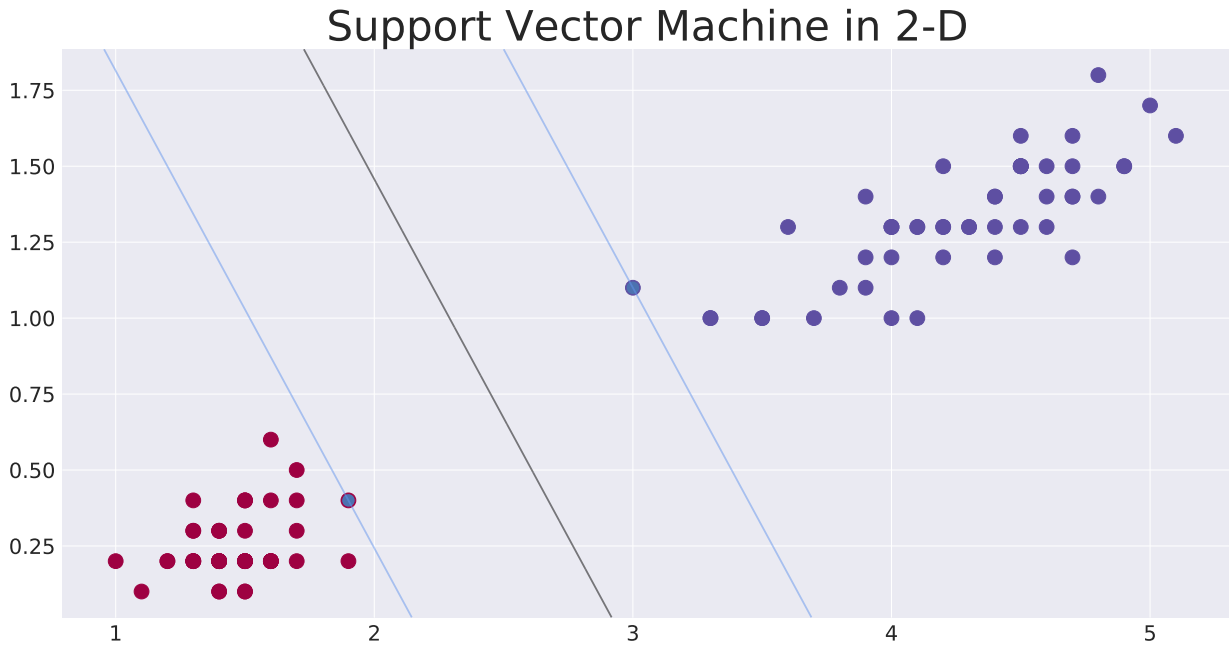


Figure 1: Support Vector Machine Classification in 2-D Example. The margins, in blue, are drawn along the data points nearest to the black hyper-plane, allowing a clean classification of the other data points. As those points are the nearest to the hyper-plane, they are the support vectors.

n-dimensional feature space, according to the following generalized function [11]:

$$\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n \tag{2.1}$$

Here, β_i is the weight corresponding to the feature x_i, and determines the hyper-plane's shape and degrees of freedom. The parameters of this generic hyper-plane are tuned, such that the perpendicular distance between differently-classified points, calculated as the dot-product between them, is maximized [12]. Formally, this separation is done according to a hyper-plane constraint:

$$y * (\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n) > 0 \tag{2.2}$$

Where, for the simple case of a binary classification, y is a function taking value one if label one applies and negative one, otherwise. Thus, to optimize the hyper-plane parameters, the SVM

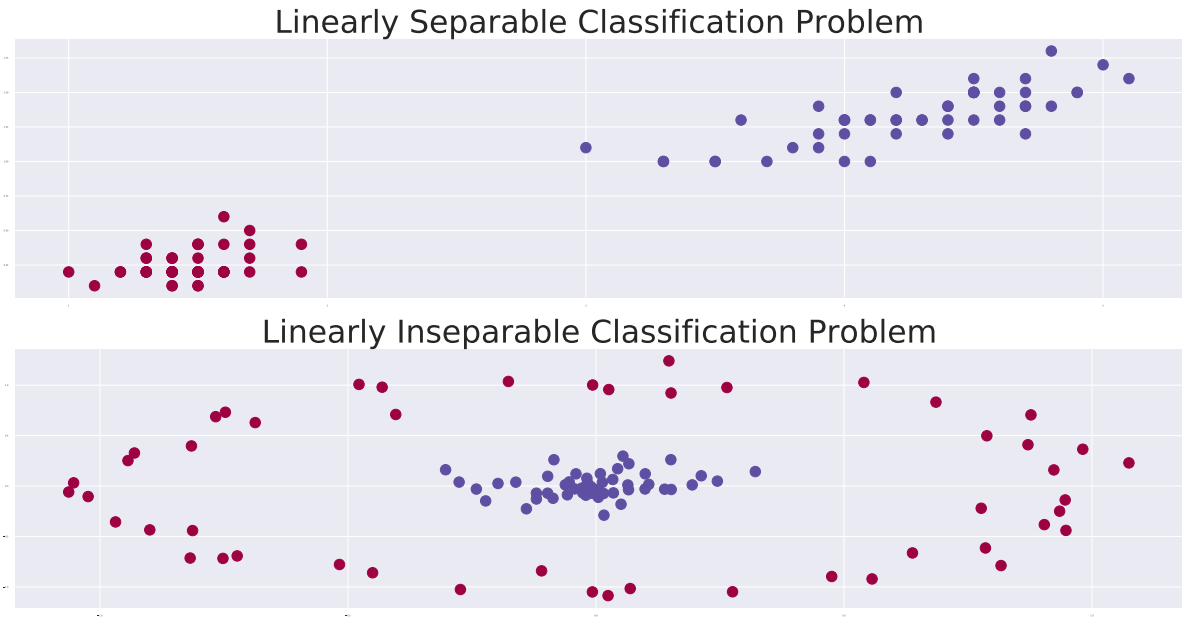


Figure 2: Linearly Separable VS Linearly Inseparable Classification Problems. Whereas in the top subplot, the distinct classes of data can be perfectly sequestered into their respective corners using a single linear hyper-plane, in the second figure a non-linear hyper-plane would be necessary to accomplish the same task.

algorithm iteratively computes the distance between each data point and the hyper-plane until it converges upon the points closest to the hyper-plane on each side. These points, the support vectors, are then the points along-which the classification margins of the algorithm are fitted. The essential features of this process are captured by the following quadratic optimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2}||\mathbf{w}^2|| + C \\ \text{s.t.} \quad & y_i(\mathbf{w}^T x_i + b) \end{aligned} \tag{2.3}$$

Where, the first line in (2.3) is the loss-function to be minimized, a measure of how close together the margins of the hyper-plane are, and the second line is the constraint specifying that no data points should lie within those margins. This quadratic programming problem is then solved algorithmically [13]. However, this procedure for modelling data is constrained to cases where the data points in each class are linearly separable (see Figure: 2); that is, a linear hyper-plane and its associated parallel margins are only capable of cleanly separating data-points of different classes,

if there is no overlap between clusters of data-points of separate classes. In practice, this is seldom the case, particularly in financial time-series. Nevertheless, the versatility of the SVM architecture lies in its ability to accommodate even linearly inseparable data (**Figure 2**) through soft-margins and/or kernel methods. Soft-margins offer a simple fix to the problem of overlapping data-points of different classes by fitting margins of the SVM hyper-plane such that some proportion of mis-classifications is allowed, subject to a pre-specified hyper-parameter C, which can be tuned by the user as needed. Thus, when hard margins are applied, C takes value 0 and no mis-classifications are allowed.

Kernel methods offer an alternative for accommodating linearly-inseparable data; by applying one of several possible data-transformation functions, such as a polynomial, a Radial-Basis, or a Sigmoid function to the original data, new features are generated along with a non-linear decision boundary, [14]. A commonly used kernel/transformation function, the Gaussian Radial Basis function, (rbf) takes the form:

$$\Phi(x, x_0) = exp(-\gamma \|x - x_0\|^2) \tag{2.4}$$

Where x_0 is the point around-which the kernel re-centers the data in its transformation and γ is a parameter modifying the width of the new non-linear margins. Though these Kernel methods provide a cleaner solution to the problem of linearly-inseparable data than "Soft Margins", selecting an inappropriate "kernel" function risks degrading the data through transformation, which can decrease the accuracy of the model.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of computational intelligence models based, albeit loosely, on the architecture of a biological human brain and the fractal structure formed by networked neuronal dendrites [1]. Unlike other models, Artificial Neural Networks are data-driven evolutionary and self-adaptive [15]; they are non-parametric learning models that rely on layers of abstraction to extract generalized relations between features and labels from data alone, without

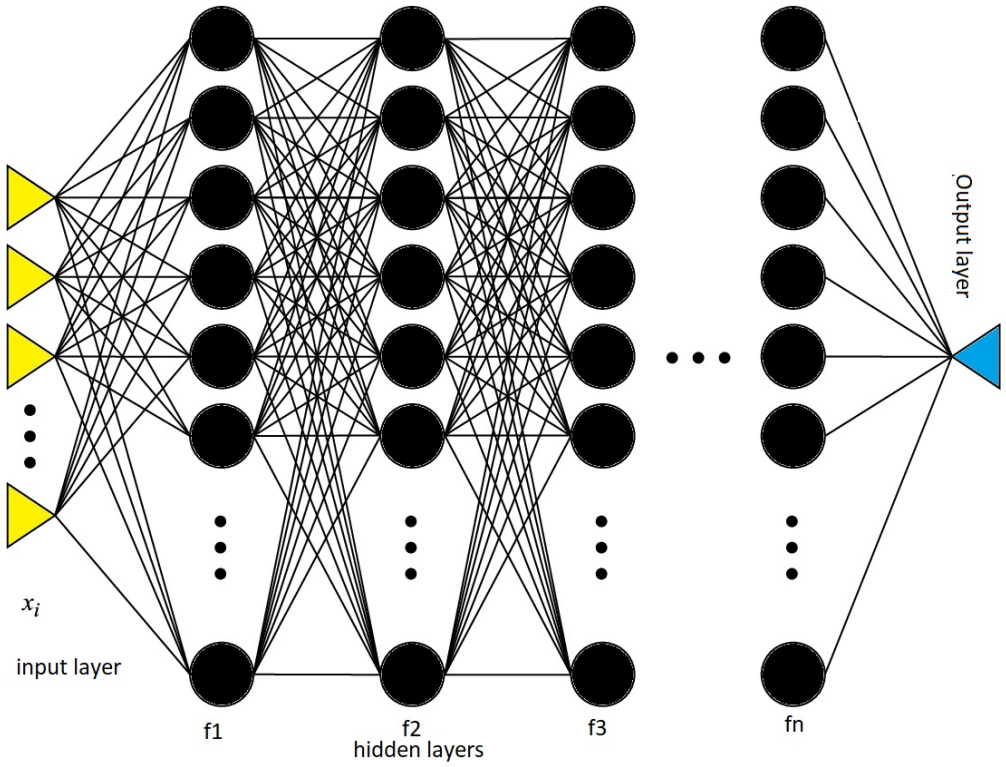


Figure 3: Illustration of the general architecture of a Feed-Forward Neural Network with an input layer, n hidden layers and an output layer (adapted from Culkin et al. (2017)) In yellow, are the input units, which feed the feature column vector, **X**, into the first hidden layer. At each hidden layer, f_j, each neuron, in black, proceeds to transform the data by applying a weight vector **W**, adds a bias, b, and then applies a pre-specified activation function, to the result. The process is done until the final output is produced in the output layer, often after transformation by a soft-max function in the nth hidden layer.

explicit human specification [7, 16]. Despite the autonomous nature of the process by which Nerual Networks "learn", the ANN architecture is, nevertheless, rich in components, many of which require initial tuning and further engineering. The optionality that each such component provides means that wide range of model architectures is housed under the umbrella class of models known as ANNs. Nevertheless, we proceed to outline the simplest case of a neural network: the Feed-Forward Neural Network (FFNN), which is also known simply, albeit inappropriately, as a Multi-Layer Perceptron (MLP).

In its most general form, the Feed-Froward Neural Network is arranged in lateral columns, called layers, that are comprised of vertically stacked nodes, called neurons, in analogy with the nodes of biological neural networks. The neurons in the first layer, the input layer L_0, take the raw data as inputs, transferring them to neurons in the first hidden layer without applying any

Name	Equation	Derivative
Identity	$f(x) = x$	$f'(x) = 1$
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
ReLU	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$
SoftPlus	$f(x) = ln(1 + e^{-x})$	$f'(x) = \frac{1}{1+e^{-x}}$

Table 1: A few commonly employed activation functions for Artificial Neural Networks

transformations or weights to the data. from there, each neuron in a given hidden layer L_j, where $j > 0$, passes all the data inputs through an activation function. The value of the activation function for each input data is then weighted by a value w_{ij}, defined by which node in the previous hidden layer, L_{j-1}, the input came from and which node of the current hidden layer it is being processed through. Since each neuron in a given layer L_j is connected to all the neurons in the preceding layer L_{j+1}, the algorithm produces an activation function a_{ij} and a weight, w_{ij} for each input. This activation function takes one of several forms, popular forms are summarized in Table 1.

Once applied to the input, the activation function allows neural networks to accommodate non-linearity within the data structure and feature-label relations. In this way, a hidden layer, L_j takes inputs from the preceding layer, L_{j-1}, transforming and restructuring them into a form from-which higher-order relations can be extracted. Thus, given an ANN with an input layer, L_0, an output layer, L_m, and $m - 1$ number of hidden layers with n neurons each, a neuron i in layer $j + 1$ transforms the data using an activation function and then applies a function of the following form:

$$z_{ij+1} = \sum_{i=1}^{n_j} w_{ij} a_{ij} + b, \qquad for \; j \in \{2, 3, ..., m-1\} \tag{2.5}$$

where the weight w_{ij} and the matrix of activation functions of the previous layer, a_{ij}, are defined as above. The resulting weighted values of the activation function values are then summed and added to a pre-specified bias value or function b. The process is then iterated across the concatenated

layers where the value of z_{ij+1} becomes the input for the activation function in the next neuron until the output layer is reached. Once reached, the final z_{im-1} is passed through a unit-step function, such as the logistic or Softmax function (**Table 1**), which converts the values corresponding to each label into a probability. Ultimately, the label with the highest probability is the one chosen as the classifier.

$$\nabla C_i(\mathbf{w}) = \left[\frac{\partial C}{\partial w_1} \; \frac{\partial C}{\partial w_2} \; \frac{\partial C}{\partial w_3} \; \cdots \; \frac{\partial C}{\partial w_n} \right]^T \tag{2.6}$$

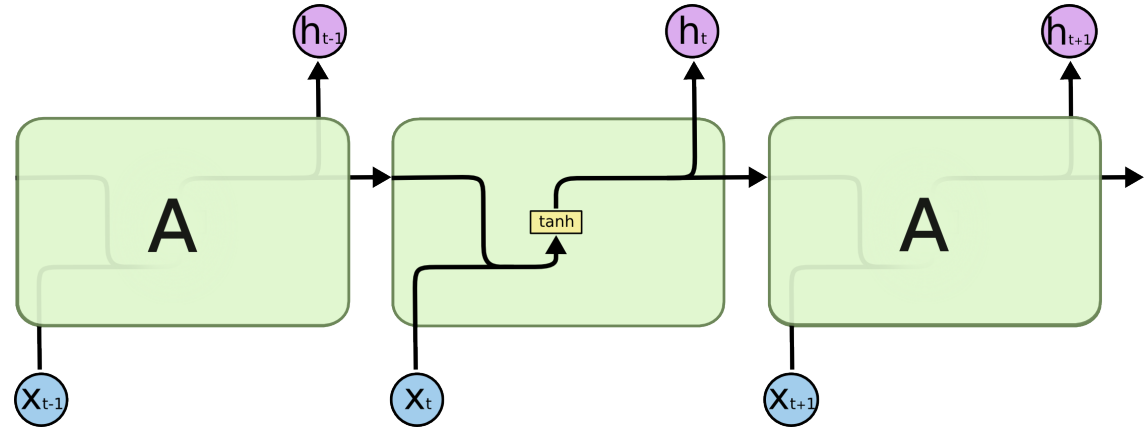
The previous paragraph outlines the general ANN testing and forecasting process. However, before this process can occur, the ANN is first trained to map relations between data by responding to the a cost-function $C_i(\mathbf{w})$. The most commonly employed method by which the algorithm "learns" is Gradient Descent (GD), where the gradient vector of the cost function, $\nabla C_i(\mathbf{w})$ (2.6), is iteratively computed and used to solve for the weight vector values $\mathbf{w}$ that minimize the loss function. Given an n dimensional weight vector, $\mathbf{w}$, for a given layer j, the weight parameter vector is updated to:

$$\mathbf{w} := \mathbf{w} - \eta \nabla C_i(\mathbf{w}) \tag{2.7}$$

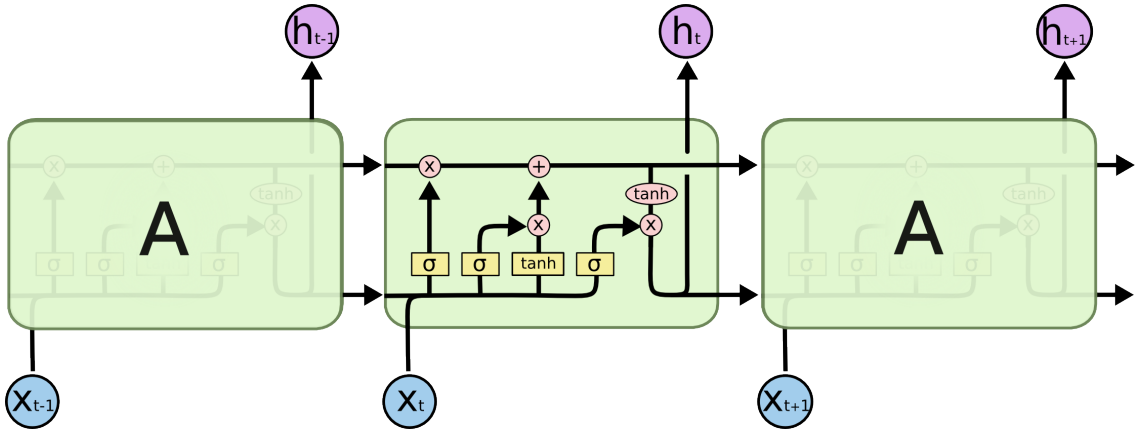
From (2.7) it is clear that by modifying the parameter η, a hyper-parameter of the algorithm, which is also known as the learning rate, the rate at which the minimization problem is solved is also changed. Likewise, since the gradient vector is a vector of partial derivatives with respect to each weight parameter (2.7), it can increase the speed at which the algorithm converges upon an optimum by specifying the direction of steepest descent on the loss function. The learning process continues and produces a model capable of assigning labels to data points to high levels of accuracy. One fundamental limitation of GD as an algorithmic programming technique is that it is likely to find only a local minimum of the loss function rather than a global one [17]. This is in contrast to the guaranteed global solutions that SVMs provide and is one advantage the former has over ANNs

[16]. Under some modifications of the learning rate, η, however, the likelihood of ANNs solving only for a local minimum can be reduced. It is also possible to minimize over-fitting, training time, and likelihood of a local optimum trap by employing Stochastic Gradient Descent (SGD) which is a modification of the classical Gradient Descent algorithm. In this form of the algorithm, an appropriate mini-batch size can be chosen to ensure that the data on which the algorithm is trained in each iteration is smaller in size and randomly shuffled.

2.2.1 Recurrent Neural Networks (RNN) and Long Short-Term Memory Neural Networks (LSTM)



(a) Neuronal Flow Chart of a Real-Time Recurrent Neural Network



(b) Neuronal Flow Chart of a Long Short-Term Memory Neural Network

Figure 4: Two diagrams capturing the fundamental differences between the neuronal operations of RNNs and LSTMs produced by Christopher Olah in 2015 [18]. The diagrams demonstrate the relative complexity of the LSTM neuronal operations that are intended to overcome the vanishing gradient problem, in both its guises, and ensure long-term dependencies are learned. Here, the yellow squares represent the layers of the network, the pink circles: Point-wise Operations,

Aside from the space of architectures allowed by varying and tuning their hyper-parameters, ANNs can differ in their fundamental shapes and neuronal operations. The result of this is a cadre of topologically distinct structures with different capacities for learning complex causal relationships from data [19]. These include Convolutional Neural Networks (CNN), used extensively in image processing problems, Generative Adversarial Neural Networks (GANN), employed on tasks where novel data need be produced that is of the same structured as the training data, and Recurrent Neural Networks (RNN) [19, 20, 21]. The topological variation that will be the focus of this research, alongside the vanilla FFNN, is the Recurrent Neural Network and, specifically, the Long-Short Term Memory Neural Networks (LSTM). Unlike the simple FFNN, Recurrent Neural Networks have the advantage of being context-aware and driven. That is, they are endowed with recurrent connections, allowing information discovered from past data to be fed back into the same neuron before being fed-forward into the next layer [21, 22]. This memory-retention property of RNNs allows them to overcome a well-known problem with FNNs, which otherwise makes it impossible to retain important information about the context of the problem [21].

While, in principle, RNNs seem perfectly fit to extracting time-dependent relationships, regardless of time-frame, they invariably fail to retain information about long-term ones, yielding networks that are unable to detect relationships across large lists/sequences of data, where relationships persist long after their inauguration. The reasons for this were poorly understood, until it was discovered that Recurrent Neural Networks, or at least the Real-time learning version of them, are susceptible to a class of problems known, generally, in the literature as ”the vanishing gradient problem” [23]. Discovered and explored by Schmidhuber and Hochreiter, this class of problems was found to result from a coupling between the iterated computation of dot-products that is integral to the back-propogation algorithm and the flow of information across the entire networks necessitated by recurrent neural networks [21]. This means that gradients computed at the network's deeper layers often have their informational implications diluted through backpropagation, such that by the time the network adjusts the weights at the shallowest layer of the network, they have either exploded or vanished. Regardless of which of these occurs, the result is a major

failure mode of Real-time RNNs. To the end of overcoming this issue, Jurgen Schmidhuber and Sepp Hochreiter, the initial discoverer of the vanishing gradient problem, developed a variant of the RNN architecture: the Long Short-Term Memory Neural Network [21].

The LSTM architecture has several components that distinguish it from a traditional Recurrent Neural Network. These difference are best captured by Figure 4. The first diagram a demonstrates the recursive and self-referential structure of Recurrent Neural Networks, in general, which takes as inputs both the outputs from iteration, h_{t-1} and current input observations from the feature columns, X_t, transforms them in accordance with a function, such as the hyperbolic tangent, and loops the output back to itself for another round. As seen in b, the structure of an LSTM is unequivocally a step in the direction of complexity. This is because the LSTM has several additional levers, called gates, allowing it to regulate, at any given time, the proportion of information retained from previous layers that is to influence what is known as the "cell state" [18]. The cell state can be thought of as the channel through-which information is linearly transformed and transmitted from one layer to the next. The introduction of gates, which channel the transformed inputs through the cell state adds complexity to this process. There are three such gates in the basic LSTM architecture.

Forget gates, f_t, are used to regulate what proportion of past information is to influence the cell state. Since the outputs of forget gates are sigmoid-activated, they take values exclusively within the interval between zero and one (inclusive). That value determines the proportion of the information to be forgotten. For clarity, the forget gate output takes the following form:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.8}$$

Where f_t is the final output of the activation layer, taking only values within the interval 0 and 1, as per the Sigmoid function's effect, h_{t-1} is the output from the previous layer, and x_t are the input observations of the current layer [18]. As in the FFNN case, W_f and b_f are, respectively, the weights and biases applied to the layer. The novel idea, here, is that, rather than taking only input observations being channeled into the current cell state, C_t, the forget layer also receives the outputs from previous cell state, C_{t-1}, as inputs.

Input gates, i_t, play two roles; the first is analogous to the role that traditional ANNs play, in that they take in current observations from feature columns, x_t, and iteratively transform them until it is known which of the hyper-parameter values are to be updated. In fact, the initial outputs of input gates are functionally the same as that of an ANN:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i \tag{2.9}$$

The second role input gates play is that of the tanh layer, which vectorizes the newly generated candidate hyper-parameters values, $\tilde{C}_t$, to be channeled into the current cell state, C_t. The values of this vector are all transformed by the hyperbolic tan function:

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C \tag{2.10}$$

The information from the input gate, and its *tanh* layer, i_t and $\tilde{C}$, are coupled to the information from the forget layer, f_t to update the previous cell state, C_{t-1} to:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.11}$$

Output gates, O_t, are the equivalent of the output layer in regular ANNs and is where the cell state is transformed for the last time before being sent to the next cell.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.12}$$

$$h_t = o_t \cdot tanh(C_t) \tag{2.13}$$

Here, the cell state is filtered through a sigmoid function followed by a tanh function; the first is done to determine which proportion of the cell state's information continues forward and the second to ensure that the outputted values are between negative and positive one. This is then outputted.

By doing this, the LSTM is able to more effectively avoid the vanishing gradient problem [21, 22].

2.3 Random Forest Algorithm as an Ensemble Method

An ensemble is a machine learning method based on aggregating the predictions and outputs of several trained models into one. The power of this agglomeration is that the space of functions it can model is expanded beyond that of any one model's [14]. One of the most successful and commonly deployed of such methods is the Random Forest (RF) algorithm. Not only do they share the versatility of ensemble methods, but the overcoming a significant short-coming of the class of its constituent models, as the outputs of RFs tend to exhibit lower variance than that of a Decision Tree [24]. Random Forests concatenate n uncorrelated decision trees that are trained on distinct sets of data, such that predictions produced by the individual models are averaged over to provide an aggregated one [14, 25]. This process is, generally, known as bootstrap aggregation or bagging, for short, and results in a robust algorithm, applicable even to complex time-series, such as those of financial markets [26]. Vitally, however, whereas regular bagging derives the feature it uses to split each node from among all features available, random forest uses only a subset of all features to determine this feature. This results in a more robust and versatile model, albeit one that is more liable to over-fit [14].

Random Forests model the relationships between features and labels by having each of their constituting decision trees sample the data, with replacement, and iteratively answer binary questions across the feature space [27]. The answers to these binary questions correspond to splits in the nodes of the trees, done until each branch is classifies members into a unique class. Tuning the hyper-parameter responsible for the depth of the trees, can increase the likelihood that the random forest will converge to such a solution. However, this is liable to yield over-fitting [24]. The other hyper-parameter Random Forest allows the user to tune, the number of trees, n, can be increased to reduce the likelihood of over-fitting. Thus, the two parameters operate at odds with each other to produce highly accurate and robust models.

2.4 Autoregressive Integrated Moving Average: a Classical Statistical Model

Unlike SVMs, ANNs, and RFs, the Autoregressive Integrated Moving Average model (ARIMA) comes from an older generation of statistical models, which dominated the realms of statistical learning and data analysis until very recently. Proposed by Box and Jenkins in the 1970s, the ARIMA(p,d,q) model was formulated to address the issues of applying traditional linear regression models to time-series data, making them highly versatile [13]. One source of this versatility are the parameters p, d, and q, which can be finely tuned according to need. It is useful to understand the parameters as relating to the ARIMA model's separable components: the Auto-Regressive (AR) and the Moving Average (MA) models. The AR constituent allows it to model deterministic, linear dependencies between its current and past values that persist across time, while accounting for stochastic disturbances, which are generated *ex-nihlo* in each period. Meanwhile, the MA component provides it with the capacity to model persistence across time of a different form. Specifically, MAs models stationary dependence across time between the current value of a series and previously occurring disturbance to that series. In general, the ARIMA(p,d,q) model takes the following form [13, 28]:

$$y_t = \theta_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + e_t - \theta_1 e_{t-2} - \ldots - \theta_q e_q \tag{2.14}$$

Here, p and q, are the auto-regressive polynomials, which account for the order of the model's auto-regressive component, ϕ and θ are the coefficients of the AR and MA components of the model, respectively, e_t, is the stochastic error term at time $t - k$, and y_t is the value of the process at time, t [29]. The parameter, d, is also significant, as it allows the user to specify, in-model, the order of differencing to which the data will be subjected [28]. However, while this modular structure makes the ARIMA model ideally fitted to modelling linear relationships encoded within data, ARIMA models are greatly diminished when applied to modelling non-linear dependencies [13]. It is important to note that the fine-tuning of the ARIMA(p,d,q) model is fundamentally predicated on the provision of stationary data series. In fact, when this condition holds, Wold's

representation theorem indicates that any stationary time-series can be approximately modelled by an ARIMA model of some order [30]. This is relevant, as it is representative of a widely noted problem within the econometric literature, where, traditionally, relationships between trending and non-stationary series were frequently found to have the verisimilitude of strongly related variables [3]. The pertinence and importance of the stationarity assumption is not the prerogative of ARIMA models, as in many cases we are interested in stationarizing the data at hand, for convenience of processing and to ensure that our algorithms are not fooled by trend correlations.

3 Financial Markets & Schools of Market Analysis

3.1 Efficient Market Hypothesis

Financial markets are the quintessential complex system, carrying information on individual and aggregate preferences, as revealed through every economic interaction and relation between the agents, institutions, and businesses embedded within them. It is, therefore, unsurprising that they have been deemed by many as opaque to prediction [13]. In fact, this belief is so well-subscribed to that it has garnered itself a name within the circles of academic and industrial economists and financiers: it is known as the Efficient Market Hypothesis (EMH). Formulated by Fama [31] in 1991, the hypothesis states that the information within financial markets is fully reflected within prices and, though authors differ on the precise implication (see [14]), it is generally thought to imply that any advantaged garnered by analyzing stock data would be arbitraged away through pricing mechanisms in financial markets [9, 24]. Largely unconvinced by the EMH, however, the vast majority of researchers working at the intersection between financial forecasting, computational intelligence, and data science continue to work to model prices and the behaviours of this system more broadly [24]. On the one hand, and by some standards, these efforts have been successful in generating learning models that successfully capture aspects of behaviours embedded within the markets [6, 5]. On the other hand, the markets *have* demonstrated their opaqueness in some sense, as the models used for studying certain components of financial markets fail to generalize to broader and different contexts. We will proceed to detail the machine learning models and papers within the literature, yielding the most promise after a discussion of the differences between

fundamental and technical analysis as mode of study.

3.2 Fundamental Analysis

The evolution of methods available for forecasting stock prices have, historically, been inextricably driven by the fundamental optimization problem underlying investing [5, 32]. that is, investors, traders, researchers, and analysts, in employing and developing myriad statistical methods, have sought to maximize their profits while minimizing the risks to which those profits, otherwise, expose them [32]. Consequently, the academic field of finance has inherited a distinction, from the financial industry, between two methodological schools of analysis, which continues to be relevant, albeit hard to dissociate from the competing dogmas that once drove it. The two schools of analysis are known as fundamental analysis and technical analysis [32].

In their stronger forms, the distinction between the two schools amounts to a difference in fundamental market epistemology. In particular, the fundamental analysts can be understood as axiomatically accepting the implications of the Efficient Market Hypothesis: namely, that the historical prices of a stock contain no useful information for forecasting the future price [32]. Instead, they propose that the key to understanding the price evolution of stocks can be found in studying the companies, markets and *fundamental* features of the economy, underlying those stocks. Historically, such features were sometimes qualitative and, therefore, harder to incorporate in any straightforward manner into models [32]. As a result, prior to the emergence of advanced learning models, such as ANNs – which have shown promise in areas such as Natural Language Processing and made sentiment-based market analysis viable, fundamental analysis was relegated to a labor-intensive task. One aspect of effective fundamental analysis required individuals to analyze reports about different companies, read through pages of news articles, and gauge general investor sentiment through conversations with other investors []. Today, machine learning models have been employed to successfully model the movement of stock prices as predicted by market sentiment in the news, on twitter, and other digital social networks [33, 34, 35, 32]. It has also been common for fundamental analysts to build models incorporating macroeconomic indicators and quantitative measures of company performance, such as price-earnings ratios, excess returns, and cash flow

[36]. Nevertheless, fundamental analysts ignore an arguably important of market analysis: historical prices and the information encoded within them.

3.3 Technical Analysis

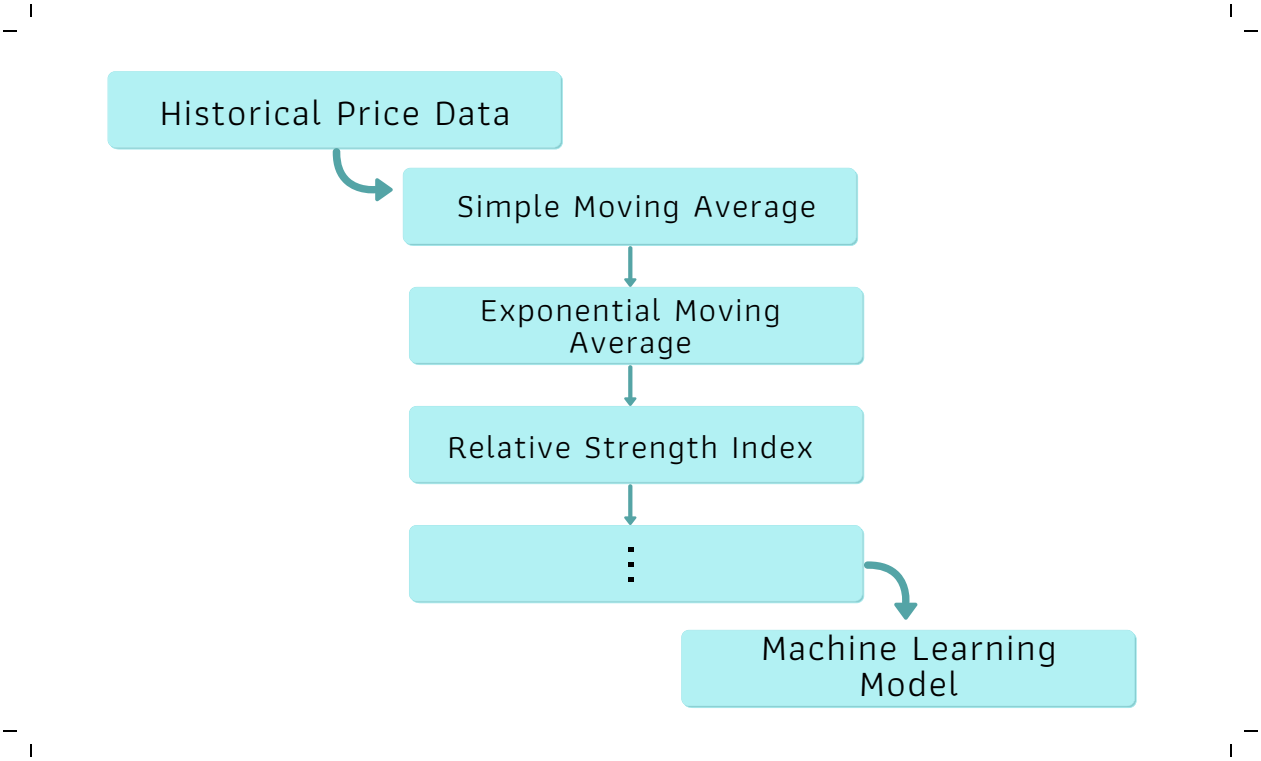


Figure 5: Flowchart of the process of Technical indicators are pre-processed and fed into the machine learning model (adapted from [32]). We have in the middle a concatenation of possible technical indicators that are produced by transformaing the historical price data. Those indicators are then fed into a machine learning model, which produces our prediction of the future price direction.

Technical analysis, unlike the aforementioned method, is associated with studying historical stock prices and their indicators to uncover patterns and information about the future. Technical analysts often employ plots of the price evolution of various stocks to extrapolate into the future [27, 37]. The advantage technical analysts have over fundamental ones is that the former's data are, invariably, structured and quantitative, which implies they are more amenable to being incorporated within statistical and computational models [32]. It is, therefore, also unsurprising that technical analysis is the more commonly employed methodology within the financial literature, with approximately 66% of 122 relevant papers, reviewed by Nti et al. in 2019, incorporating it [32]. In the following research we will primarily be using technical analysis and technical indicators, as they

require transformations of different aspects of the prices. For example, technical indicators, such as the Relative Strength Index (RSI), On-Balance-Volume (OBV), simple moving and exponential moving averages (SMA & EMA), and Moving Average Convergence/Divergence (MAC/D), and Stochastic Oscillator (see Figure: 5) [32, 24]. We will proceed to outline how we compute each of these indicators and what they measure in Section 8.2 of the paper.

4 Support Vector Machine Applications to Financial Forecasting

4.1 Financial Forecasting using SVMs

Support Vector Machines rank as the second most commonly deployed method in the financial forecasting literature [5]; this is due to the versatility, robustness, and computational inexpensiveness of the method, which, unlike Artificial Neural Networks, is capable of handling noisy, high-dimensional, and small data-sets [6]. In fact, the authors [38] of the most cited article within the Financial Forecasting literature, according to Henrique [5], purported to have successfully employed Support Vector Machines to forecast Korean stock price movements. Though the results of this paper likely inspired a wave of publications in the financial forecasting literature using SVMs, there is evidence that the results were dubious [5]; however, three years later, Manish and Thenmozhi [25] used the same indicators used by Kim [38] as input variables, albeit applied this time to *S&P* 500 directional movements, to demonstrate the superiority of SVMs to Random Forests, ANNs, and other traditional models. Building on this work, a paper demonstrating the potential of Support Vector Machine for producing robust forecasts of volatile markets was produced two years later by Yang, Chan, and King [39]. Gradually, these applications were generalized; in 2005, Huang, Nakamori, and Wang [40] published an article detailing applications of a Support Vector Classifier to the forecasting of the Nikkei 225 stock index's movements. Today, this paper serves as one of the seminal papers in the literature, as it was one of the first to compare the performance of Artificial Neural Networks with Support Vector machines on a classification problem and to propose an efficacious Hybrid SVM methodology [5]. The results of the comparison indicate that the accuracy of the SVM is the highest, out-competing even the Artificial Neural Network [40, 5].

Multiple papers are published over the years following this that attempt to compare performance

across tasks between neural networks and support vector machines of various types [41, 42, 43, 44]. One, published in 2011 [41] compares the performance of a standard Support Vector Machine with a standard Artificial Neural Networks. The two models are parametrized by data from over 10 years of daily prices with The aim to test which one will classify the daily directional movements of the ISE National 100 Index with a higher accuracy [5]. The experimental set up this comparison as determined by the volatility of the Istanbul Exchange, an emerging market, and the abundance of data, makes it a test of both robustness against volatility and over-fitting. The results indicate that the Artificial Neural Network has a superior predictive capacity, significantly outperforming the standard SVM model. Notably, a comparison between SVM, Random Forest, K-Nearest Neighbor (KNN), Naive Bayes, and Soft-max Neural Networks produces similar results, with SVM performing badly. In this case, however, Random Forest performs best [45].

In contrast with the prior results, comparisons such as those conducted by Sheta [43], and Das and Padhy [42] have resulted in SVMs performing best; in the first case, this can be explained by the implementation of an SVM with a radial Basis Function Kernel to account for non-linearity and linearly inseparable data. The utility being able to account for non-linearity in financial data is liable to increase prediction accuracy significantly. In the second case, however, the higher level of performance is liable to be best accounted for by the Back-propagated Neural Networks over-fitting and lacking generality [42].

4.2 SVM Hybrid Methods for Financial Forecasting

Ensemble bagging methods, also known as Hybrid Methods, are commonly used in Financial Forecasting application of machine learning to improve the performance of an algorithm. Individual machine learning model types differentially perform depending on the type of data-set they are applied to and have idiosyncratic drawbacks, such as over-fitting and parameter-tuning requirements. It is, therefore, often hard to determine which singular method is appropriate to apply to a given task and how it should be configured such that it performs optimally. To bypass the obstacle of having to select one model at the exclusion of all others, an alternative class of machine learning algorithms, known as ensemble methods, have been proposed [46].

This class of methods permits combining multiple machine-learning methods, which reduces parameter uncertainties and performance volatility [47]. In 2005, Pai and Lin [48] implemented a Hybrid Model, combining SVMs with Auto-regressive Integrated Moving Average Models (ARIMA) to forecast stock prices, as opposed to merely stock price movement directions. This offered an insight into the capacity for achieving more robust performance by coupling different methodologies. This set a precedence for coupling traditional statistical techniques with machine learning algorithms, which as a result became more common in the literature; In 2009, a two-stage architecture was proposed by Hsu et al. [49], this linking self-organizing feature mapping (SOFM) to support vector regression. In the first stage the SOFM was applied to the feature space, mapping data points from where they are onto a new space where they would be clustered based on shared statistical properties. This would aid in capturing the non-stationarity of financial series; the Support Vector Regression would then take the transformed data and use them as inputs for financial forecasting with significant success in predicting seven stock indices. Analogously, Huang and Tsai [50] later applied the same hybrid model successfully to predict the price changes of an index on the Taiwanese exchange, indicating the robustness of the model [5].

Alternatively, but similarly in two-stage structure, researchers began stacking multiple models of the same architecture in an attempt to enhance performance; models such as the Genetic Algorithm and SVR hybrid, implemented by Huang [51] and the SVR-SVR model, implemented by Nayak et al. [52] were developed. In these models, the first-stage model would be used as a pre-processing and/or feature selection method for the input data that would produce a transformed data-set to be used as input for the second-stage model. In the second stage, the forecasting process would proceed as normal but the results were notably better [5].

4.3 Optimized SVMs for Financial Forecasting

SVMs can, alternatively, be finely tuned using optimization, which involves using pre-processing methods or tuning the architecture of the model before forecasting. In the case of SVMs, the straightforward Kernel Methods built into it are considered a form of optimization method. Thus, SVMs with polynomial and radial basis function kernels are, generally, more efficacious at predict-

ing directional stock price change [41]. Another form of optimized Support Vector Machines is the Localized Support Vector Classifier, an innovative solution to the problem of accommodating local volatility within the global solutions produced by regular kernel-based SVMs [53]. This is applied to ensure that locally volatile financial data points do not distort classification. There are other such optimization methods that have been frequently applied within the literature to combat the failure modes SVMs are susceptible to [5].

5 Artificial Neural Network Applications to Financial Forecasting

5.1 Financial Forecasting Using Artificial Neural Networks

The introduction of Neural Networks in 1943 [1] was left, largely, untapped for decades until the development of the modern computer, with its multi-transistor processors and enhanced RAM, made the iterated, simultaneous, and calculus-based optimization required by ANNs with multiple layers and nodes (now called Deep Neural Networks) feasible to do at reasonable time-scales [1]. Since then, various authors made the observation that Artificial Neural Networks have become the preferred models for financial forecasting [9, 6, 5]. The flexibility of the activation functions, which allow ANNs to extract non-linear relations from complex and noisy data, provides them with an unrivaled capacity for handling stochastic dynamical systems such as stock prices [54]. Neural Networks were first applied to the field of finance in 1989 [5]. The paper, published by White [55] described an implementation of Feed Forward Neural Networks (FFNNs) to abstract patterns from the seemingly stochastic evolutionary pathways of asset prices. While the idea behind Neural Networks has existed for long, its efficacy has only recently become comparable, and in fact superior, to other statistical learning methods. Since then, researchers have shown the capacity for neural networks to out-perform alternative machine learning methods at any statistical or financial problem [56]. Leading one of the earliest investigation into financial time series and NNs, Garliauskas [57] applied a Kernel-modified Neural Network to predicting financial time-series. Two years after White's seminal paper, Leung et al. [58] examined the difference in accuracy between classification Neural Networks, such as the Probabilistic Neural Network, and Level-estimation Neural Networks, and the Multilayered Feed-Forward Neural Networks, which had thus-far been

applied, on stock indices. The results derived from a test on stock indices provided evidence that models for predicting directional changes were more efficacious than models forecasting specific price values. Chen et al. [59] took the idea of a Probabilistic Neural Network further by testing it on predictions about the emerging markets in Taiwan. Aside from the increased accuracy provided by only forecasting price direction, The benefits of harnessing a Probabilistic Neural Network is that learning is Bayesian and therefore both quicker and able to deal with outliers [59].

As neural networks were applied to multiple alternative Financial applications, their susceptibility to initial tuning was discovered to both allow a great level of versatility and also to cause the network's performance to vary significantly [60]. In 2004, Thawornwong and Enke [61] attempted to explore how the predictions of neural networks change depending on the input values. By adjusting each of the weights of the input values to correspond to the amount of information each added to the model, the authors found that dynamic variable selection per period acted as a diversification strategy, hedging against risk and allowing more accurate predictions to be made across time. In 2005, to further explore the degree to which Neural Networks were out-performing structural economic linear models Cao, Leggio, and Schniederjans [62] compared the capital pricing model (CAPM) and Fama's 3-factor model to multivariate and uni-variate neural networks. Unsurprisingly, the results showed that neural networks outperformed the linear models on multiple measures, including mean squared error and accuracy rate.

5.2 Hybrid ANN methods for Financial Prediction

Aside from the standard Artificial Neural Network Computational Models, which allow a wide variety of variance depending on training technique, initially assigned weights, and activation functions, Hybrid Artificial Neural Network Models have been commonly employed in the financial forecasting literature [63, 14]. Similarly to Support Vector Machine Hybrid Methods, ANN-based Hybrid Methods allow combinations of Machine Learning models to be used. The financial Forecasting literature is replete with various examples of ANN Hybrid Models; in some applications, Neural Networks have been combined with classical parametric models [64, 63]; a few notable examples are outlined here.

In 2010, Wu and Shahidehpour [63] applied a tripartite method for forecasting electricity prices. In the first step, the authors fitted an auto-regressive moving average time-series model with exogenous features (ARMAX) in order to estimate the relationship between differently-lagged price returns and various features. The residuals resulting from this initial step were then used to estimate a Generalized Auto-regressive Conditional Heteroscedasticity (GARCH) model, which was ultimately used to train a Neural Network with wavelet activation functions (AWNN) capable of learning the non-linear relations of the data. Similarly, Wang et al. [64] combined binomial trees, finite differences and Monte Carlo simulation with a linear Neural Network and a Support Vector Machine to predict option prices, finding that this increased the accuracy of predictions, as compared to previous option price prediction models.

With the development of ensemble methods, complementary models could be used in conjunction to outperform others. In 2008, Yu, Chen, Wang, and Lai proposed precisely this type of model [65]. The model combined Support Vector Machines with a genetic algorithm for feature selection; the authors built an evolutionary Least Squares Support Vector Machines (LSSVM) using a Genetic Algorithm (GA). This model had the benefits of low computational cost, and an already-high capacity for generalization augmented by an algorithm for engineering optimal values at each iteration of the learning process [65]. The forecasting results of this model outperformed Neural Networks trained by Back-propagation along with various other models. Henrique [5] notes that the Yu et al [65] apply McNemar's test to compare model results, indicating some level of robustness.

5.3 Optimization Methods for Financial Forecasting with ANNs

Optimization techniques have been applied to Neural Networks in various ways to ease the parameter-tuning requirements. Genetic Algorithms, in particular, are commonly applied to select relevant features, set up an appropriate number of hidden layers and to optimize predictive performance as a result [66]. In 2000, to forecast Stock Price Index 200 on the Korean Exchange, Kim and Han [67] coupled a Neural Network with a genetic algorithm. The performance of this model in predicting weekly directional changes in the market was over 80%. In the same year as Kim and Han, Phua et al. [68] successfully used the same ANN optimization ensemble to forecast the movements

of a Singaporean stock exchange index with an accuracy of over 80%, suggesting some degree of robustness to the result.

6 Random Forest Applications to Financial Forecasting

As an ensemble machine learning method that is accommodated by several programming languages [69], Random Forests have played a role in pushing the adoption of modern machine learning techniques and, particularly ensembles in various fields, with finance being no exception [69]. Applications of Random forest within finance, have been implemented in areas from fraud detection,to forecasting banking failure and stock prices [69, 70, 71]. In the area of financial forecasting, the advantage of Random forest derives from its afore-mentioned and explored versatility; the model's two parameters allow for a wide array of tuning, while not imposing on the user complexity of having multiple hyper-parameters, such as those of a neural network, to adjust and tinker with. In 2 [26], Random Forests are applied to forecasting the stock price movements of the SP 500 and compared with models, such as neural networks and gradient-boosted trees. In that application, Krauss et al. find that Random Forest outperforms both neural networks and gradient boosted trees. Aside from such comparative applications, Random Forest models have been concatenated with other models to build larger and more powerful ensembles. Recently, Kumar et al. [72] employed an ARIMA-Random Foret hybrid, alongside their ARIMA-SVM, mentioned above. Similarly, Ballings et al., deploys various models, including a random forest, SVMs, KNN, and ANN and investigates which hybrid or ensemble model between them is best at forecasting stock prices [27]. These implications, while not belabouring the the point, make it clear that RFs are a commonly applied and often viable model for financial applications.

7 Software & Hardware

The entirety of our research, from data pre-processing and feature extraction, to exploratory data analysis, model implementation, and evaluation is conducted in Python 3.8, using packages, such as *Numpy*, for fast linear algebra operations *Pandas*, for its easy data manipulation. We deploy our Random Forest Model and Support Vector Classifier using *Scikit-learn* and utilize Google's *Tensorflow* library, with its high-level API, *Keras*, to develop both our Deep and Long Short-Term

Index	Symbol	Country of Exchange	Closing Time (EDT)
S&P 500 Price Index	GSPC	United States	16:00
Dow Jones Industrial Average	DJI	United States	16:00
NYSE Composite Index	NYA	United States	16:00
NASDAQ Composite	IXIC	United States	16:00
Russell 2000 Index	RUT	United States	16:00
FTSE 100	FTSE	United Kingdom	11:30
CAC 40	FCHI	France	12:00
DAX Performance Index	GDAXI	Germany	11:30
Hang Seng Index	HSI	China	04:00
Nikkei 225	N225	Japan	02:00

Table 2: A list of information about the indices we retrieve for analysis, including each index's name, symbol, the country of the exchange on-which each index is traded, and the closing time of that exchange.

Neural Networks. Furthermore, we implement our models in the *Google Colab* environment using their free GPU and CPU clusters. Finally, we provide our code fully open source on Git-Hub [73].

8 Methodology

Our methodology can be broken down into five steps, the first of which is retrieving the data for our selected indices from Yahoo Finance and compiling them into a *.CSV* file. From there, we proceed to the second step, that of exploratory data analysis; here, we investigate the data by plotting and calculating summary statistics to check correlations and attributes, such as whether they are stationary. The third step, feature extraction & pre-processing, is where we clean our data, generate the technical indicators to be used as features, and define target variables, for the purpose of training and testing our models. Once we have such a cleaned data-set, constituted exclusively of those labels and features, we proceed to the fourth step, in which we split the data into training and testing periods. Fifthly and finally, we implement our four models and proceed to evaluate their performance according to two measures, accuracy and F-Measure.

8.1 Data Compilation

We derive the structured historical stock data of interest, namely those of ten international stock indices, from Yahoo Finance. In particular, we download daily data on five US indices: the S&P 500 Price Index (GSPC), Dow Jones Industrial Average (DJIA), the NYSE Composite Index (NYA),

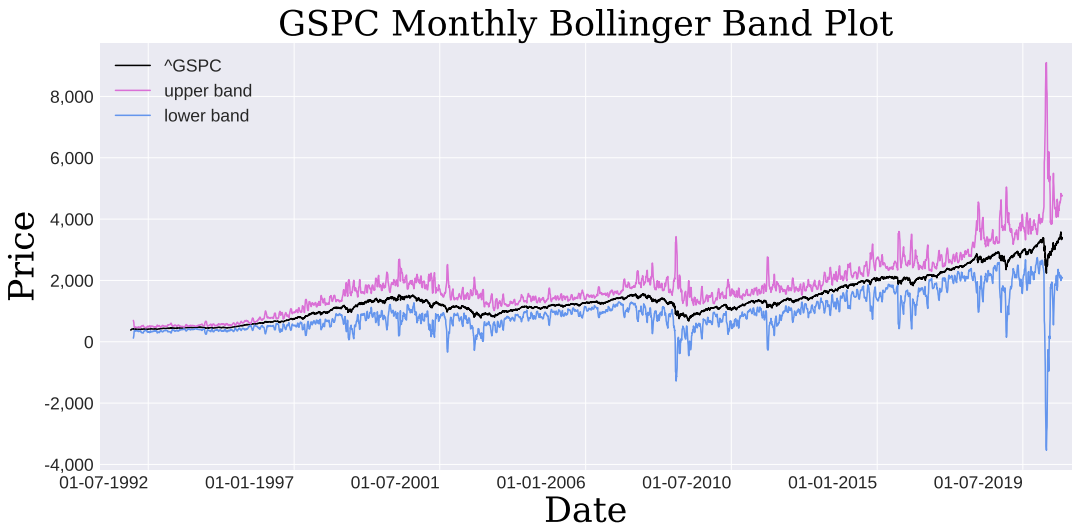


Figure 6: Plot of 21-Day Bollinger-Bands: one of multiple simple technical indicators we generate for use in the data analysis.

the Russell 2000 (RUT), and the NASDAQ Composite Index (IXIC), three European indices: the FTSE 100 (FTSE), CAC 40 (FCHI), and the DAX Performance Index, and two Asian indices: the Hang Seng Index (HSI) and the Nikkei 225 (N225). Our data cover the period from 15/12/1991 to 15/09/2020. Furthermore, we derive data for the VIX, the main index used as an implied volatility measure, over the same time period to compute values, such as the volatility simple and exponential moving averages. These data will be the source of our technical indicators. We avoid using additional macroeconomic or fundamental indicators to i) more readily compare results across the various models and ii) so that our results are comparable to other papers within the literature that employ technical methodologies [26].

8.2 Feature Extraction & Pre-processing

8.2.1 Technical Indicator Generation

We begin by using the historical adjusted closing prices data to compute simple technical indicators for the GSPC adjusted closing prices, such as the 3-day, 5-day (weekly), 10-day (bi-weekly), and 21-day (monthly) simple and exponential moving averages (MA03, MA05, MA10, and MA21), Bollinger-Bands for the same periods, rolling maxima and minima, and others in classical manner

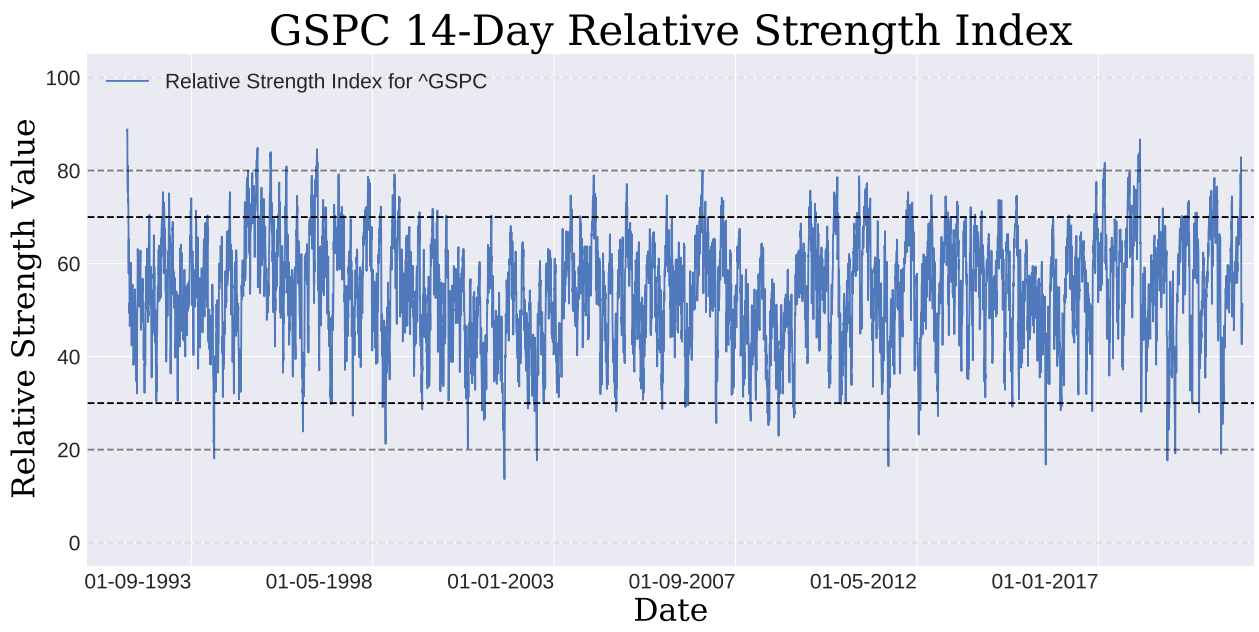


Figure 7: Time Series Plot of 14-Day Relative Strength Index. We have plotted the momentum indicator, along with intervals showing the threshold levels under-which it is justified to predict a downward or upward turn in the price. We can see that the, notably, there are exceptionally sharp troughs near both the time of the 2008 crisis and that of the recent events of COVID-19

[26, 24, 14] (see Figure 6). Furthermore, we compute more complex technical indicators, measuring momentum, volume, and other such characteristics of the GSPC prices [14, 26, 45, 74]. In what follows, we define the various non-trivial technical indicators and the rationale for using each one.

The Relative Strength Index (RSI) is the first of such advanced indicators to be computed, and is used to assess whether a stock is being over-bought or over-sold (see Figure 7). Specifically, the RSI measures whether a stock is over or under-priced, relative to the one such technical indicator we engineer for the purposes of forecasting the returns of several stock indices is the Relative Strength Index. The Relative Strength Index is a measure of the degree to which a given security is inappropriately priced and can be thought of as a comparison between recent price changes and

the ratio of average gain to loss over the period n. In particular, the RSI is calculated as:

$$RSI_n = 100 - \frac{100}{1 + rs_n} \tag{8.1}$$

$$rs_n = \frac{average\ gain_n}{average\ loss_n} \tag{8.2}$$

Where n is the "look-back" period under consideration, for which the average gain or loss will be computed. Given the nature of the RSI formula, its values can be interpreted as percentages, with a value of 70 or above indicating that the asset is comparatively over-bought, which means it is liable to fall down in the near-future (hence, sell) and, conversely, a value of 30 or below being an indicator that the price is about to increase (hence, buy).

In our particular case, we are interested in a standard momentum indicator and so compute the 14-day RSI. This should be significantly correlated with the direction of our returns. Of course, there are other momentum indicators that we compute, to ensure that we are appropriately hedged against the possible drawbacks of each individual measure of stock price momentum.

We also compute the **Moving Average Convergence/Divergence (MACD)** trend-momentum indicator. This indicator compares the trend of a 12-day exponential moving average with that of a 26-days moving average by taking the difference between them and generate a signal line based on their 9-day exponential moving average. The signal line, acts as an indicator that the market is over-pricing or under-pricing an asset according to whether its trend has an upward steep slope or downward one. This offers an alternative to the RSI, which, rather than following a smoothed or de-noised trend of the plot, instead examines the changes in daily price, as compared to recent gains and losses.

We also compute the **Stochastic Oscillator (SO)** and **William's Percent-R (WR)**, which are often used as alternatives. The former, is calculated according to the following formula:

$$\%K = (\frac{C - L_n}{H_n - L_n}) * 100 \tag{8.3}$$

Where, C, the recent closing price, is subtracted from L_n and H_n are, respectively, the lowest

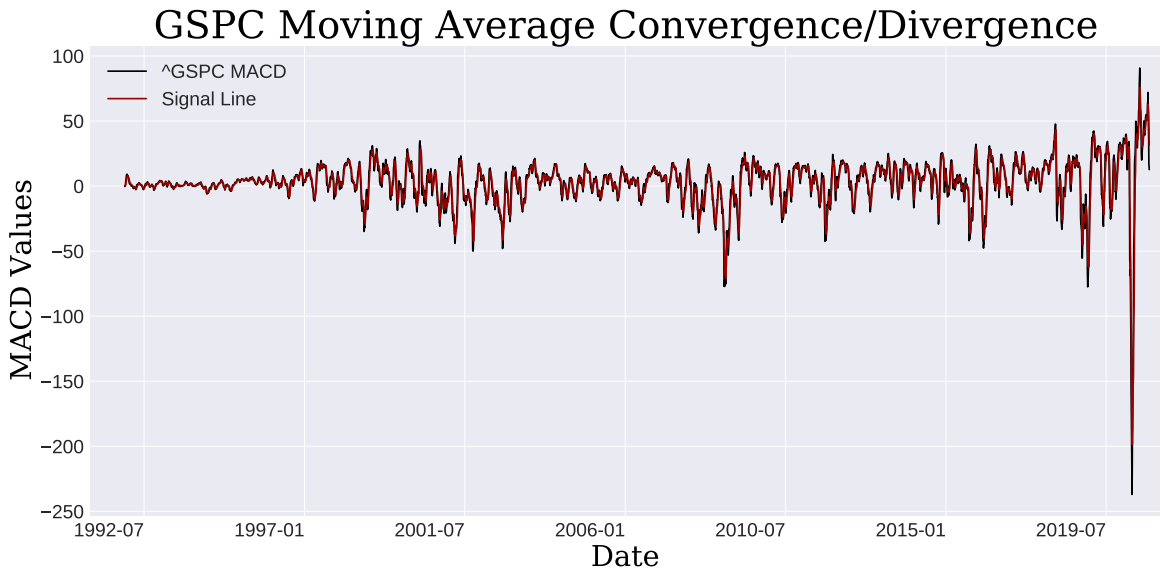


Figure 8: Plot of Moving Average Convergence/Divergence Indicator & Signal Line. Hwere we can see that, the signal line acts as a de-noised version of the MACD, allowing predictions about the momentum to be made beforehand. The indicator is,importantly, unbounded and, therefore, is able to capture well sharp under-pricing, such as in 2020, as a result of COVID-19 and in 2008/9, where a sharp trough occurs, likely as a result of the financial crash.

and highest stock prices over the n previous trading days, usually 14. We then proceed to take the 3-period moving average of, %K, to generate the value to be used as the SO. This value is always bounded and so can, once again, be thought of as a percentage. Here, values over 80% are considered indicators of an overbought stock, whereas values below 20% are indicators of the opposite. The SO is meant to be used within smaller trading windows, which means it is less dynamic than the other measures of momentum. William's %R is an alternative to the SO and, thus, behaves very similarly, except that its bounding values are changed from 0 and 100 to -100 and 0. The negative values maintain the meanings of their corresponding positive counterparts. However, the indicator is calculated as follows:

$$W\%R = \frac{Highest\ High_n - Close}{Highest\ High_i - Lowest\ Low_i} \tag{8.4}$$

Where the values are self-explanatory here and n is, as before, the "look-back" window.

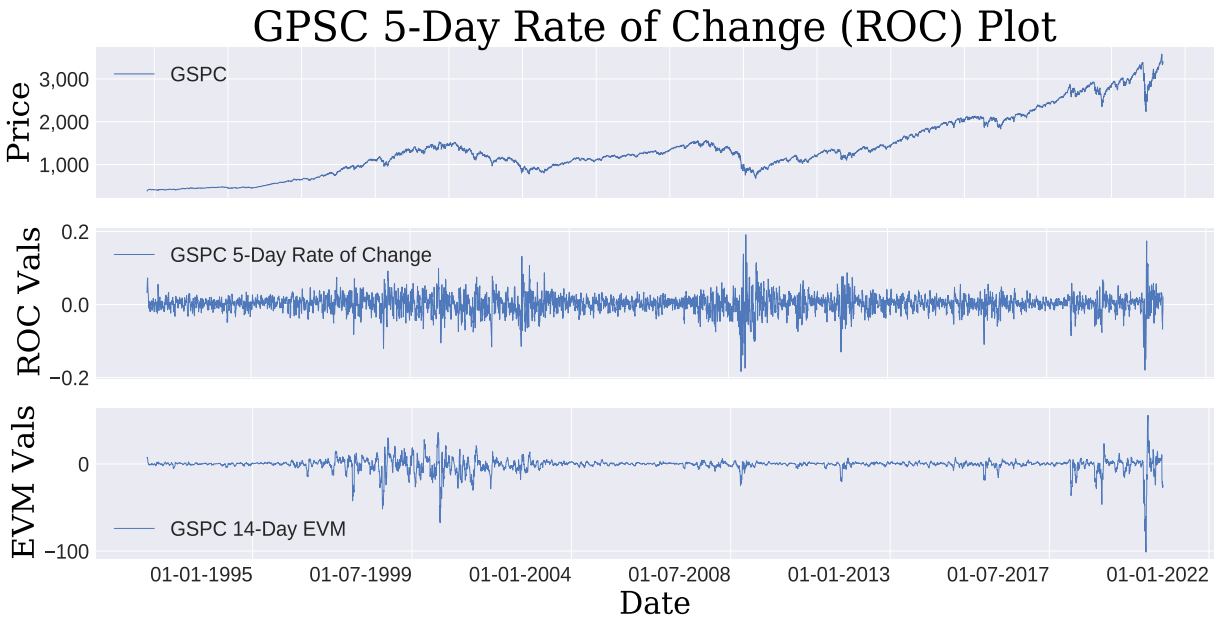


Figure 9: Time Series Plot of GSPC closing prices, 5-Day Rate of Change (ROC), and 14-Day Ease of Movement (EVM). Plotted along with the closing prices of the GSPC, we see that the ROC and EVM values, while similar, capture different aspects of the plot, with the ROC being less sensitive to the magnitude of the changes and more sensitive to the direction, relative to the EVM.

The final momentum measure we compute is the **Price Rate of Change Ocillator (ROC).** This indicator is computed in accordance with the following formula:

$$ROC = (\frac{Closing\ Price_p - Closing\ Price_{p-n}}{ClosingPrice_{p-n}}) * 100 \tag{8.5}$$

where n is as above and p is the most recent time. The ROC, as its name suggests, is an unbounded measure, centred at zero, of the change in price over a given window. Fortunately, the ROC is versatile, as by modifying n, traders can adjust their momentum measure for a more accurate picture of any time-span, short or long-term. We are interested in providing our models with a view of both the weekly and monthly returns and so we compute the ROC with $n = 10$ and $n = 21$.

In addition to momentum indicators, we compute a volume indicator, to measure the strength behind a stock price oscillation. That indicator is known as the **Force Index (FI):** an unbounded

index that fluctuates between positive and negative values. It can be used for understanding how significant a trend or trend change is; in particular, the FI can be used to detect significant inflection points in the market. It is computed in accordance with the following formula:

$$FI(1) = (Closing\ Price_t - Closing\ Price_{t-1}) * VFI(13) \tag{8.6}$$

Where VFI(13) is a 13-day exponential moving average of FI(1). We compute this, alongside, what is known as, the **Ease of Movement Indicator (EVM)**, which captures both aspects of the momentum and the volume in one indicator. We calculate the values of this indicator as follows:

$$[]D = (\frac{High_t + Low_t}{2} - \frac{High_{t-1} + Low_{t-1}}{2})BR = (\frac{\frac{Volume}{Scale}}{High_t - Low_t}) \tag{8.7}$$

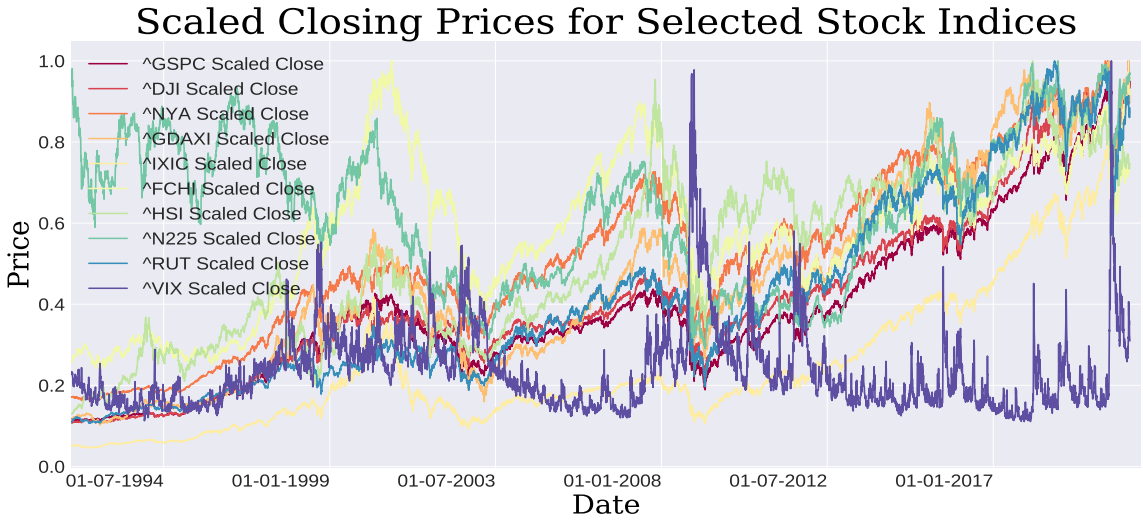
Where D is the distance moved by the stock as measured by the difference between the averages of the maximum and minimum at each point in time, t, and its preceding point, $t-1$, and BR is the Box Ratio measuring the ratio between the volume of traded stocks and the difference between high and low.

8.2.2 Pre-processing of Stock Indices

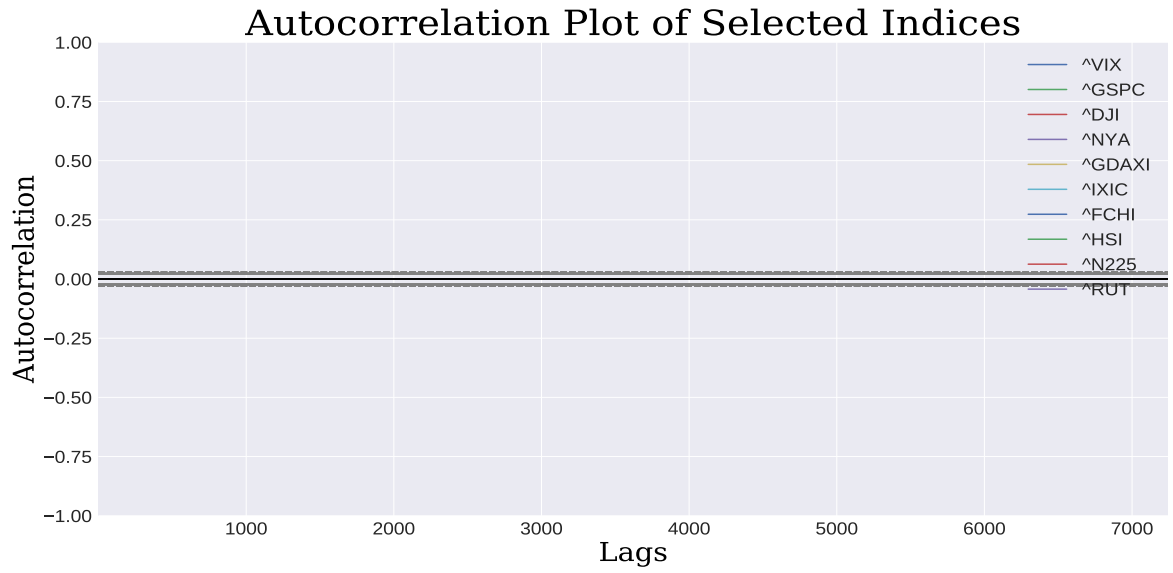
Once we have generated the appropriate technical features based on our data, we proceed to re-code our stock indices for the purposes of forecasting forecasting. To provide this research with an appropriate level of realism, we assume that, when forecasting the GSPC price, at time t, we have only at our disposal the realized prices of the stock up to and including yesterday, $t-1$. Furthermore, as the GSPC is available on the US stock exchange, we assume that we only have data available on US Stocks up until and including yesterday; these conditions are necessary if we are to acquire realistic results. For non-US indices, however, it is acceptable to have prices up until and including today, as, at least for the indices with-which we are concerned, their adjusted closing price for time t will be available before that of GSPC is. Thus, we transform our data, accordingly, lagging all US index data by 1 lag and keeping the rest of the index data as they are.

From there, we consider the importance of lagged terms, by examining the correlation between GSPC and lagged versions of itself. Finding significant correlation that recedes only by the third lag in non-US indices and the fourth in US Indices, we generate two data vectors, one of single lags of the data and another of double lags. Finally, drop the missing values that have appeared at the top of our data, such that we now have a cleaned data-set.

8.3 Exploratory Data Analysis



(a) Time-Series Plot of Scaled Closing Data



(b) Auto-Correlation Plot of Closing Data

Figure 10: Two diagrams capturing the essential features of the time-series. The data are evidently non-stationary, as signalled by the trending behaviour of the time-series plot and the non-receding auto-correlation between lags in the auto-correlation plot.

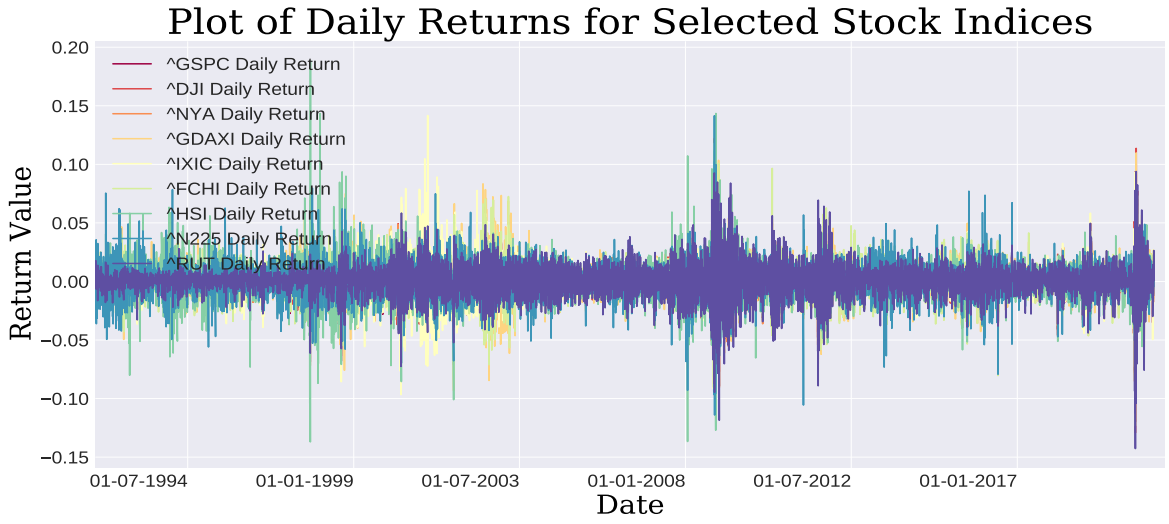
To begin investigating the data, since the prices of our closing data are at different levels, we plot the scaled adjusted closing prices, which we obtain as follows:

$$SP_t = \frac{P_t}{P_{max}} \tag{8.8}$$

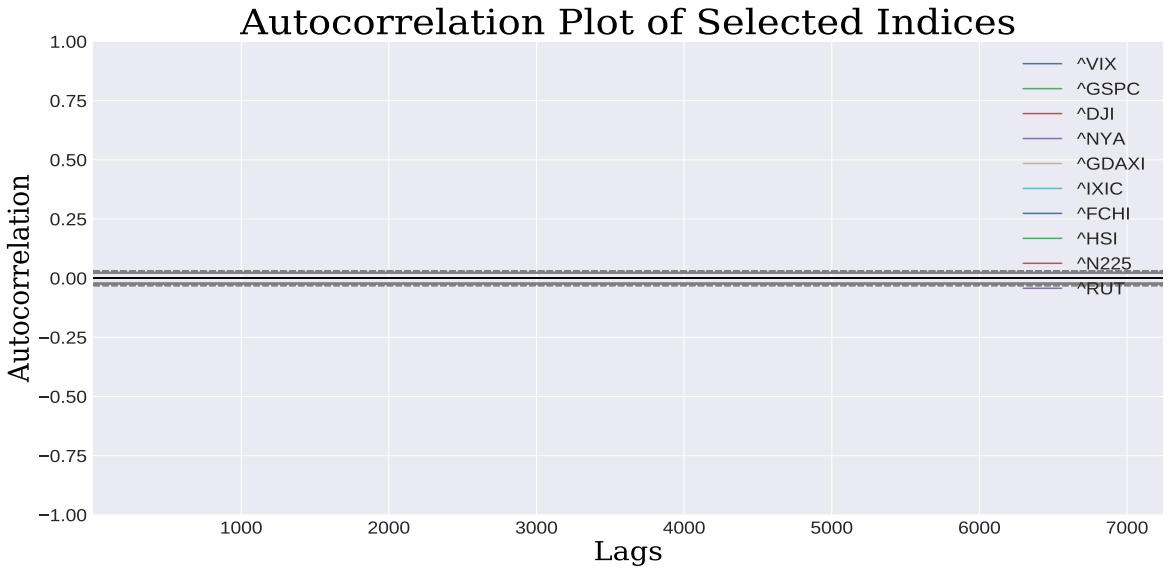
Where SP_t, the scaled price, is generated by dividing the price at time, t, by the maximum closing price. This makes it so that our data are uniformly scaled and correlations between individual indices are easier to spot in the plot (see Figure: 10a. We can see that there are strong positive correlations between the indices' closing prices; in particular, the US stocks tend to move tightly together, with the Asian and European stocks trailing, albeit not by much. Furthermore, we see that peaks in the VIX, the implied volatility measure, as expected, tend to precede disturbances to and oscillations in the trends of prices. Finally, the plot hints at the fact that our data are non-stationary, as their constituent stocks are, on the whole and with the exception of the VIX, strictly trending upwards 10a. We later generate our target value for our model implementations as the directional movements of the GSPC daily returns.

To further examine the stationarity of our series, we examine the auto-correlation plot of the data and find that it indeed confirms the hypothesis that the data are non-stationary 10b. We proceed, with what is standard practice, to compute the daily returns according to this formula and examine the resulting auto-correlation and time-series plots to see whether the data are stationary. As attested to by the constant means and variances in Figure 11a and the barren auto-correlation plot in Figure 11b, whereas the adjusted close prices of the data were not, their daily returns are stationary. For the purposes of better understanding the data, we plot the histograms for the various index returns (see Figure . It is apparent from the histograms that the returns tend to, generally, follow a bell-shaped curve. This is deceptively similar to a gaussian; however, upon examining kurtosis, we find that all of them exhibit lepto-kurtic tails. That is, the data generating process of the daily returns qualitatively differs as from that of a Gaussian distribution, in that extreme or tail-event occur more frequently. Notably, the GSPC is the most lepto-kurtic of all of the indices.

Beyond the stationarity, we proceed to examine the correlation matrix and kernel density esti-



(a) Time-Series Plot of Daily Returns Data



(b) Auto-correlation Plot of Daily Returns

Figure 11: Two diagrams capturing the essential features of the time-series. The data are stationary, as signalled by the constant means and variances of the time-series plot and teh absence of apparent correlation between lags in the auto-correlation function plot.

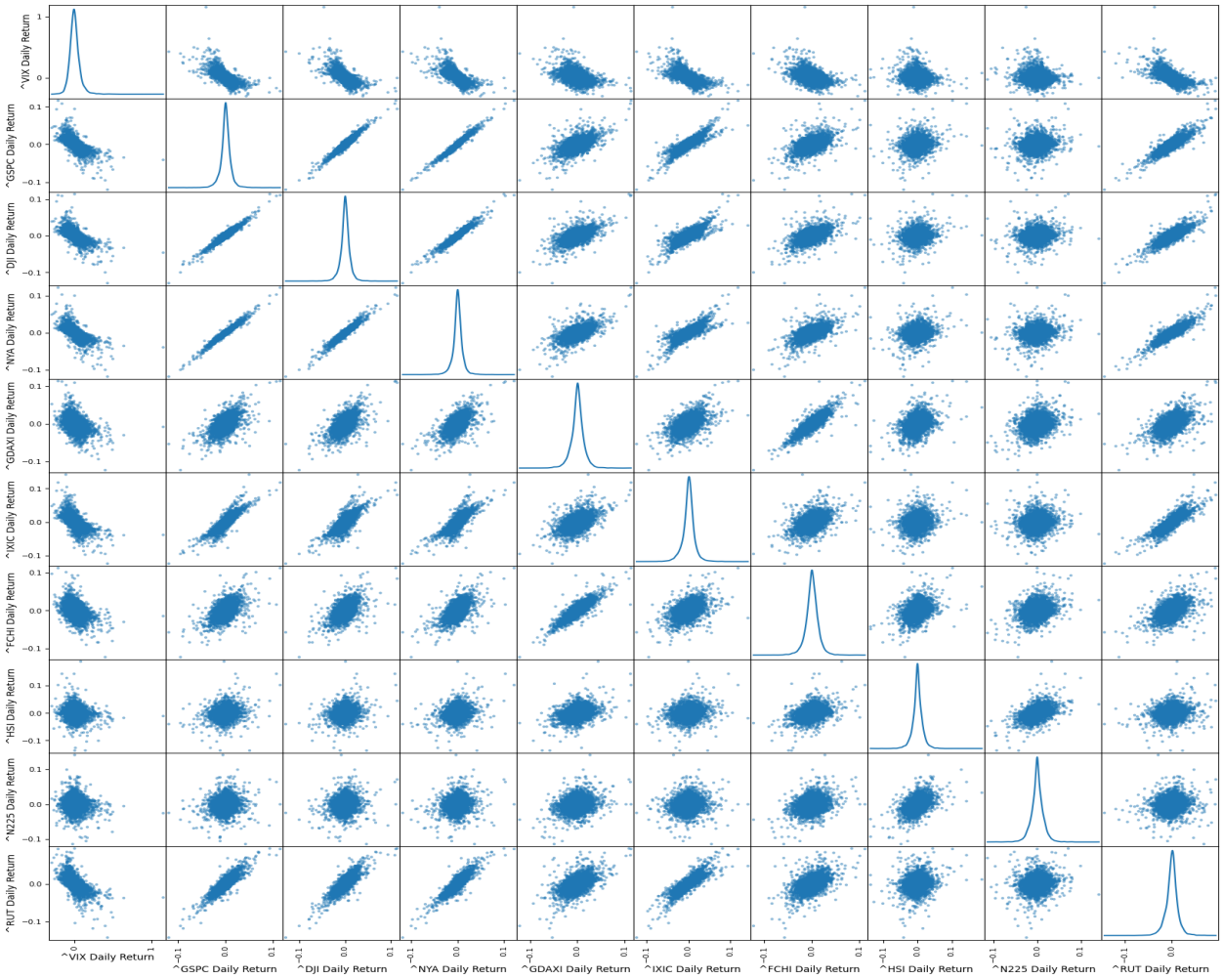


Figure 12: Scatter Correlation Matrix We can see strong positive correlations between the five US Stock Indices and weaker, but still strong, positive correlation between the European stock indices. However, the two Asian stock indices seem uncorrelated

GSPC Return Direction Corrrelations	
Variable	**Pearson Correlation**
GSPC 5-day Force Index	0.17
GSPC 14-day William's R%	0.37
GSPC 14-day Ease of Movement	0.07
GSPC 14-day Relative Strength Index	0.28
GSPC 14-day Stochastic Oscillator %D	0.11
GSPC 14-day Stochastic Oscillator %K	0.37

Table 3: Only these features are corr elated at higher than 5% with GSPC Return Direction, according to a simple Pearson correlation. We do not include the other variables in the table, but retain them in the features data, so as to make use of them during model implementation, as non-linear models may prove capable of extracting relationships from them.

mations of the data, using a scatter plot matrix (see). The kernel density estimations reveal what we already understood from the histograms about the densities of the indices. As for the correlations. We can see that there are clear and tight positive correlations between four of the five US indices (GSPC, DJI, NYA, and RUT), with the fifth (IXIC) being positively correlated with the other indices, albeit not as tightly. Unsurprisingly, we see a clear and tight positive correlation between our two European indices (GDAXI and FCHI). However, there appears to be no clear relationship between our two Asian indices and any other index. Fortunately, there are slight correlations between GSPC and the European Indices. These correlations are promising, as they indicate that our non-technical indicators serve a purpose. However, it is vital that we see the technical indicators.

Analysis of the Pearson correlation coefficients between our technical indicators and the GSPC Return Direction, suggests that there are six measures that are in-fact highly correlated (see Figure . This concludes the second step of exploratory data analysis.

8.4 Training & Testing Split Methodology

To prepare it for the analysis, we take our data, which now ranges from 16/01/1992 to 15/09/2020, and split it into a training and testing period. We use the first 90% of our data, covering , as our training data and the remaining 10% as our testing data. This is done to ensure our models have enough training examples relative to the data they are being tested on. From there we further divide

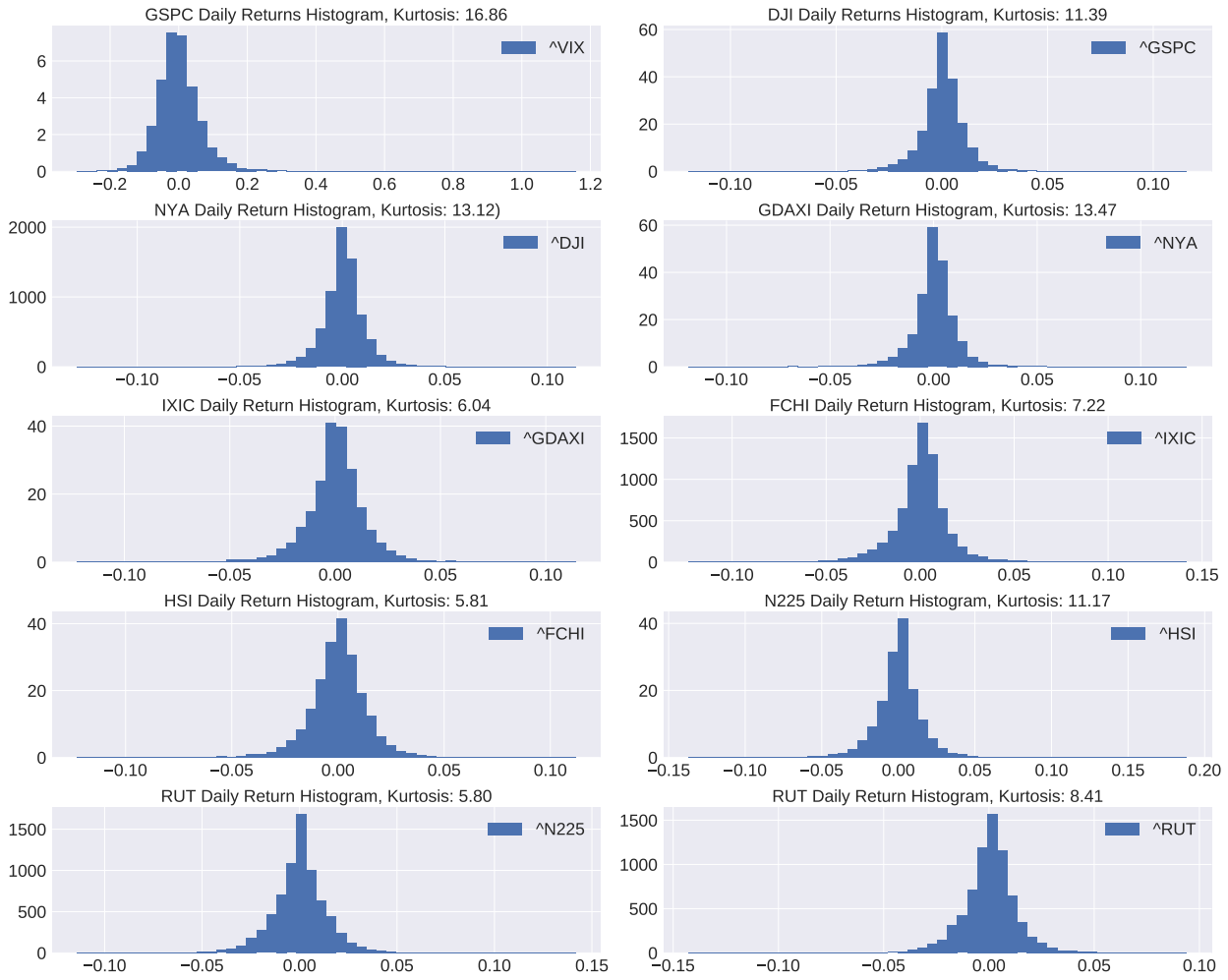


Figure 13: Histogram of Daily Returns for Each Stock Index. We can see that, although the daily returns have the verisimilitude of a Standard Gaussian distribution, this belies their underlying "fat-tails" or kurtosis levels

Table 4: Model Implementation & Results

Machine Learning Model	*Method*	*Recall*	*Precision*	*Accuracy*	*Test F-Measure*
Random Forest w. XGBoost	XGBoost	84.5%	83.4%	82.0%	83.9%
Feed-Forward Neural Network	Adam Optimizer	100%	55.8 %	55.8%	71.63 %
Support Vector Classifier w. rbf	rbf-kernel	76.6%	58.9%	63.9%	71.9%
Long Short-Term Memory Neural Network	Adam Optimizer	77.1%	59.5	63.7%	66.7%

Note: We use the Adam as the optimizer and MSE as the loss function for both the LSTM and FFNN, for a cleaner comparison. Furthermore, we set the learning rate for both models at 0.02.

our data into our features, **X**, and our target, y. This will allow us to consecutively train the model on one set of data and then evaluate its performance using another. Moving forward. We proceed to implement our models in this way.

8.5 Model Implementation, Evaluation & Discussion of Results

We implement each of the models and use the Precision, Accuracy, and F-Measure, computed as follows, to evaluate relative model performances:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{8.9}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{8.10}$$

$$Accuracy = \frac{TruePositives + TrueNegatives}{TruePositives + TrueNegatives + FalsePositives + FalseNegatives} \tag{8.11}$$

$$F - Measure = 2(\frac{Recall * Precision}{Recall + Precision}) \tag{8.12}$$

We can see the results, as measured in this way, in table

We implement a **Random Forest with XGBoost** and tune its parameters by running multiple

iterations of different parameter variations, until we converge upon an optimal setup of 80 trees, each of which has a maximum depth of 8. Furthermore, we consistently increase the learning rate, starting at 0.0001, until we converge upon 0.115 as an appropriate learning rate for the model. As observed in the results table, the Random Forest with XGBoost outperforms all other models on all three metrics used for analysis . We implement a **Support Vector Machine with RBF Kernel**, after testing the performance of polynomial kernel SVCs of various degrees and the linear SVCs, only to find that the RBF-Kernel is superior in this application. The result is that it is the second best performing of our models, scoring right under the Random Forest on precision, accuracy, and the F-Measure. We then proceeded to implement the neural networks using the Keras-TensorFlow API and, though they were far from optimally tuned, the Adam optimizer, which automatically adjusts the learning rate based on the frequency with which different parameters are being updated, allowed for a demonstration of the vast discrepancies between a traditional FFNN and a Long Short-Term Memory Neural Network, which is manifested in the results.

Our traditional **Feed Forward Neural Network** was composed of 3 hidden layers, an input layer, and a Soft-max output layer. In particular, we found that 3 hidden layers, with 1000 neurons in the first layer, 500 in the second, and 10 in the third, provided the maximum power possible, while remaining computationally feasible. For these layers, we applied the popular and preferred "ReLU" activation, in the hidden layers. Of course, in the FFNN's case, there were no dropout values or memory cells and, therefore, it was no surprise that the performance was poor. Since we are using it as a benchmark, this is not a problem. However, the **Long Short-Term Memory Neural Network**, which was built of two hidden layers and an output layer performed significantly better. The two layers consist of 10 neurons, with 0.2 dropout rate at each layer, allowing the LSTM to forget 20% of the data after each layer. The results are surprising, albeit in line with previous work by [22]. Thus, while LSTM neural networks have shown promise in other applications, such as that conducted by Krauss and Fischer [26], they are, in this case, dominated in performance by both the Random Forest and Support Vector Classifier. However, as there is much work that could have been done in the direction of further tuning the architecture of the LSTM, these results

serve only as preliminary data rather than any firm confirmation of a hypothesis. Nonetheless, it is vital that more data be garnered through future research in a structured manner that serves to build up a theory regarding the types of problems with-which the different models are best-equipped, respectively, to deal.

9 Conclusion

This article has endeavoured to survey the literature on Random Forests, Support Vector Machines, Neural Networks, and their associated ensemble methods; it summarizes the key architectures implemented within the financial forecasting literature and provides insight into the developments in the area, concerning the three best-performing models within the field. What the literature bears out is that, although the field of financial forecasting is budding with advanced techniques and leveraging complicated ensemble methods, it is lacking the rigor required to develop a deeper, full-scale view of how the methods at hand can be improved or measured against a singular and unified standard. In particular, it is clear that there is a need for replication and bench-marking, as encouraged by the works of Krauss and Fischer [26], as well as Patel et al. [14]. Methods that are developed, should be applied to different tasks, with varying data-sets, and within a wide array of contextual environments. Far from this, at the moment, the literature is, instead, replete with applications of new methods to the same tasks and data-sets or, worse, novel cases, for-which no existing bench-mark previously exists. By exposing the flaws of this literature's structure the aim is to have paved a way for measuring new methodologies and advancements against more rigorous scientific standards.

The paper proceeds from this analysis of the literature to provide an implementation and examination of the three most well-established machine learning algorithms to the single task of forecasting the direction of GSPC daily returns. Moreover, it compares these three against a well-established benchmark, the FFNN model and uses the same features and labels to train and test the models, comparing them on the metrics of accuracy and F-Measure. The results bear out that, in this case, the best performing methods are the XGBoosted-Random Forest, followed by the LSTM, then the SVM, with radial-basis-function Kernel, and, finally, the FFNN, which provides

our benchmark in this case. These results, while preliminary and contingent on the task, data window, and stock index used, serve to pave the way for a more unified approach to bench-marking and comparing machine learning methods to financial applications. We hope to see future research explicate the same practices and provide more data on the comparative performance of different models on other financial tasks and/or in other environments.

References

[1] Culkin R. Machine learning in finance : The case of deep learning for option pricing. 2017.

[2] Cesa M. A brief history of quantitative finance. *Probability, Uncertainty and Quantitative Risk*. 2017, 2(1):6.

[3] Ventosa-Santaulària D. Spurious regression. *Journal of Probability and Statistics*. 06 2009, 2009. https://doi.org/10.1155/2009/802975. Available from: `https://www.hindawi.com/journals/jps/2009/802975/`.

[4] Ferson WE, Sarkissian S, and Simin TT. Spurious regressions in financial economics? *The Journal of Finance*. 2003, 58(4):1393–1413. Available from: `http://www.jstor.org/stable/3648215`.

[5] Henrique BM, Sobreiro VA, and Kimura H. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*. 2019, 124:226–251. https://doi.org/10.1016/j.eswa.2019.01.012.

[6] Cavalcante RC, Brasileiro RC, Souza VL, Nobrega JP, and Oliveira AL. Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*. 2016, 55:194–211. https://doi.org/10.1016/j.eswa.2016.02.006.

[7] Atsalakis GS and Valavanis KP. Surveying stock market forecasting techniques – part ii: Soft computing methods. *Expert Systems with Applications*. 2009, 36(3):5932 – 5941. https://doi.org/10.1016/j.eswa.2008.07.006.

[8] Cortes C and Vapnik V. Support-vector networks. *Machine learning*. 1995, 20(3):273–297. https://doi.org/10.1007/BF00994018.

[9] Yoo PD, Kim MH, and Jan T. Ensemble methods in machine learning. multiple classifier systems. In: *2005 Pakistan Section Multitopic Conference, 2005, London, UK*, p. 1–7. London: Springer, 2005. https://doi.org/10.1109/INMIC.2005.334420.

[10] Hua S and Sun Z. Support vector machine approach for protein subcellular localization prediction. *Bioinformatics (Oxford, England)*. 2001, 17:721–8. http://doi.org/10.1093/bioinformatics/17.8.721.

[11] Khemchandani R, Jayadeva , and Chandra S. Regularized least squares fuzzy support vector regression for financial time series forecasting. *Expert Systems with Applications*. 2009, 36:132–138. http://doi.org/10.1016/j.eswa.2007.09.035.

[12] Xu Q, Zhou H, Wang Y, and Huang J. Fuzzy support vector machine for classification of eeg signals using wavelet-based features. *Medical Engineering Physics*. 2009, 31:858–865. https://doi.org/10.1016/j.medengphy.2009.04.005.

[13] Pai PF and Lin CS. A hybrid arima and support vector machines model in stock price forecasting. *Omega*. 2005, 33(6):497–505. https://doi.org/10.1016/j.omega.2004.07.024.

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

[15] Zhang P, Patuwo E, and Hu M. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*. 03 1998, 14:35–62. http://doi.org/10.1016/S0169-2070(97)00044-7.

[16] Xu Q, Zhou H, Wang Y, and Huang J. Fuzzy support vector machine for classification of eeg signals using wavelet-based features. *Medical Engineering Physics*. 2009, 31:858–865. https://doi.org/10.1016/j.medengphy.2009.04.005.

[17] Gheyas IA and Smith LS. A novel neural network ensemble architecture for time series forecasting. *Neurocomputing*. 2011, 74(18):3855–3864. https://doi.org/10.1016/j.neucom.2011.08.005.

[18] Olah C. Colah's blog, 2015. Available from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[19] Liu W, Wang Z, Liu X, Zeng N, Liu Y, and Alsaadi FE. A survey of deep neural network architectures and their applications. *Neurocomputing*. 2017, 234:11–26. https://doi.org/10.1016/j.neucom.2016.12.038. Available from: `https://www.sciencedirect.com/science/article/pii/S0925231216315533?via%3Dihub`.

[20] Alqahtani H, Kavakli-Thorne M, and Kumar G. Applications of generative adversarial networks (gans): An updated review. *Archives of Computational Methods in Engineering*. 2019, p. 1–28. https://doi.org/10.1007/s11831-019-09388-y. Available from: `https://link.springer.com/article/10.1007%2Fs11831-019-09388-y`.

[21] Hochreiter S and Schmidhuber J. Long short-term memory. *Neural computation*. 1997, 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735. Available from: `https://www.bioinf.jku.at/publications/older/2604.pdf`.

[22] Fischer T and Krauss C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*. 2018, 270(2):654–669. https://doi.org/10.1016/j.ejor.2017.11.054. Available from: `https://www.sciencedirect.com/science/article/pii/S0377221717310652?via%3Dihub`.

[23] Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*. 04 1998, 6:107–116. 10.1142/S0218488598000094. Available from: `https://www.bioinf.jku.at/publications/older/2304.pdf`.

[24] Basak S, Kar S, Saha S, Khaidem L, and Dey SR. Predicting the direction of stock market prices using tree-based classifiers. *The North American Journal of Economics and Finance*. 2019, 47:552–567. https://doi.org/10.1016/j.najef.2018.06.013. Available from: `https://arxiv.org/pdf/1605.00003.pdf`.

[25] Kumar M and M. T. Forecasting stock index movement: A comparison of support vector machines and random forest. *SSRN Electronic Journal*. 2006. https://doi.org/2139/ssrn.876544.

[26] Krauss C, Do XA, and Huck N. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the sp 500. *European Journal of Operational Research*. 2017, 259(2):689 – 702. ISSN 0377-2217. https://doi.org/10.1016/j.ejor.2016.10.031. Available from: http://www.sciencedirect.com/science/article/pii/S0377221716308657.

[27] Ballings M, Van den Poel D, Hespeels N, and Gryp R. Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications*. 2015, 42(20):7046 – 7056. ISSN 0957-4174. https://doi.org/10.1016/j.eswa.2015.05.013. Available from: http://www.sciencedirect.com/science/article/pii/S0957417415003334.

[28] Choi HK. Stock price correlation coefficient prediction with arima-lstm hybrid model. *arXiv preprint arXiv:1808.01560*. 2018. Available from: https://arxiv.org/pdf/1808.01560v5.pdf.

[29] Ariyo AA, Adewumi AO, and Ayo CK. Stock price prediction using the arima model. In: *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, p. 106–112, 2014. Available from: https://ieeexplore.ieee.org/abstract/document/7046047.

[30] Rose DE. Forecasting aggregates of independent arima processes. *Journal of Econometrics*. 1977, 5(3):323 – 345. ISSN 0304-4076. https://doi.org/10.1016/0304-4076(77)90043-4. Available from: http://www.sciencedirect.com/science/article/pii/0304407677900434.

[31] Fama EF. Efficient market hypothesis. *The Journal of Finance*. 1991, 46:383–417. http://doi.org/10.2307/2325486.

[32] Nti IK, Adekoya AF, and Weyori BA. A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*. 2019, p. 1–51. https://doi.org/10.1007/s10462-019-09754-z. Available from: https://link.springer.com/content/pdf/10.1007/s10462-019-09754-z.pdf.

[33] Li X, Xie H, Chen L, Wang J, and Deng X. News impact on stock price return via sentiment analysis. *Knowledge-Based Systems*. 2014, 69:14–23. https://doi.org/10.1016/j.knosys.2014.04.022. Available from: https://www.sciencedirect.com/science/article/pii/S0950705114001440.

[34] Makrehchi M, Shah S, and Liao W. Stock prediction using event-based sentiment analysis. In: *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, p. 337–342, 2013. https://doi.org/10.1109/WI-IAT.2013.48. Available from: https://ieeexplore.ieee.org/document/6690034.

[35] Hagenau M, Liebmann M, and Neumann D. Automated news reading: Stock price prediction based on financial news using context-capturing features. *Decision Support Systems*. 2013, 55(3):685 – 697. ISSN 0167-9236. https://doi.org/10.1016/j.dss.2013.02.006. Available from: http://www.sciencedirect.com/science/article/pii/S0167923613000651.

[36] Richardson S, Tuna I, and Wysocki P. Accounting anomalies and fundamental analysis: A review of recent research advances. *Journal of Accounting and Economics*. 2010, 50(2):410 – 454. ISSN 0165-4101. https://doi.org/10.1016/j.jacceco.2010.09.008. Available from: http://www.sciencedirect.com/science/article/pii/S0165410110000406.

[37] Wei LY. A hybrid anfis model based on empirical mode decomposition for stock time series forecasting. *Applied Soft Computing*. 2016, 42:368 – 376. ISSN 1568-4946. https://doi.org/10.1016/j.asoc.2016.01.027. Available from: http://www.sciencedirect.com/science/article/pii/S156849461630014X.

[38] Kim KJ. Financial time series forecasting using support vector machines. *Neurocomputing*. 2003, 55:307–319. https://doi.org/10.1016/S0925-2312(03)00372-2.

[39] Yang H, Chan L, and King I. Support vector machine regression for volatile stock market prediction. volume 2412, p. 391–396, 2002. https://doi.org/10.1007/3-540-45675-9_58.

[40] Huang W, Nakamori Y, and Wang S. Forecasting stock market movement direction with support vector machine. *Computers OR*. 2005, 32:2513–2522. https://doi.org/10.1016/j.cor.2004.03.016.

[41] Kara Y, Boyacioglu MA, and Baykan OK. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert Syst. Appl.* 2011, 38:5311–5319.

[42] Das S and Padhy S. Support vector machines for prediction of futures prices in indian stock market. *International Journal of Computer Applications*. 03 2012, 41:22–26. 10.5120/5522-7555.

[43] Sheta A, Ahmed S, and Faris H. A comparison between regression, artificial neural networks and support vector machines for predicting stock market index. *International Journal of Advanced Research in Artificial Intelligence*. 2015, 4:55–63. https://doi.org/10.14569/IJARAI.2015.040710. Available from: https://www.researchgate.net/publication/280045818_A_Comparison_between_Regression_Artificial_Neural_Networks_and_Support_Vector_Machines_for_Predicting_Stock_Market_Index.

[44] Jain N, Kumar S, Kumar A, Shamsolmoali P, and Zareapoor M. Hybrid deep neural networks for face emotion recognition. *Pattern Recognition Letters*. 2018, 115:101–106. https://doi.org/10.1016/j.patrec.2018.04.010. Available from: https://www.sciencedirect.com/science/article/pii/S0167865518301302?casa_token=mNG9nv8JJhsAAAAA:

O4GzAeQLM8c8tHFY-lEpWHTPKbQ7SmZZrRIYCnGJtXxkfbuzPsCb5PdBePG7Om_rgUZ_0iRj1g.

[45] Kumar I, Dogra K, Utreja C, and Yadav P. A comparative study of supervised machine learning algorithms for stock market trend prediction. In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, p. 1003–1007, 2018. https://doi.org/10.1109/ICICCT.2018.8473214. Available from: https://ieeexplore.ieee.org/document/8473214.

[46] Schwaighofer A and Tresp V. The bayesian committee support vector machine. In: *International Conference on Artificial Neural Networks*, p. 411–417. Springer, 2001. https://doi.org/10.1007/3-540-44668-0_58. Available from: https://link.springer.com/chapter/10.1007/3-540-44668-0_58.

[47] Kourentzes N, Barrow D, and Crone S. Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*. 07 2014, 41:4235–4244. 10.1016/j.eswa.2013.12.011. Available from: https://www.researchgate.net/publication/259165043_Neural_Network_Ensemble_Operators_for_Time_Series_Forecasting.

[48] Pai PF and Lin CS. A hybrid arima and support vector machines model in stock price forecasting. *Omega*. 2005, 33(6):497–505. https://doi.org/10.1016/j.omega.2004.07.024. Available from: http://jasonarrabito.com/Hybrid.pdf.

[49] Hsu SH, Hsieh JPA, Chih TC, and Hsu KC. A two-stage architecture for stock price forecasting by integrating self-organizing map and support vector regression. *Expert Systems with Applications*. 2009, 36(4):7947–7951. https://doi.org/10.1016/j.eswa.2008.10.065. Available from: https://www.sciencedirect.com/science/article/pii/S0957417408007860.

[50] Huang CL and Tsai CY. A hybrid sofm-svr with a filter-based feature selection for stock market forecasting. *Expert Systems with Applications*. 2009, 36(2):1529–1539. https://doi.org/10.1016/j.eswa.2007.11.062. Available from: https://www.sciencedirect.com/science/article/pii/S0957417407006069.

[51] Huang CF. A hybrid stock selection model using genetic algorithms and support vector regression. *Applied Soft Computing*. 2012, 12(2):807–818. https://doi.org/10.1016/j.asoc.2011.10.009. Available from: https://www.sciencedirect.com/science/article/pii/S1568494611004030?via%3Dihub.

[52] Nayak RK, Mishra D, and Rath AK. A naïve svm-knn based stock market trend reversal analysis for indian benchmark indices. *Applied Soft Computing*. 2015, 35:670–680. https://doi.org/10.1016/j.asoc.2015.06.040. Available from: https://www.sciencedirect.com/science/article/pii/S0169207099000485.

[53] Yang H, Huang K, King I, and R. L M. Localized support vector regression for time series prediction. *Neurocomputing*. 2009, 72(10):2659 – 2669. ISSN 0925-2312. https://doi.org/10.1016/j.neucom.2008.09.014. Available from: http://www.sciencedirect.

com/science/article/pii/S0925231208004529. Lattice Computing and Natural Computing (JCIS 2007) / Neural Networks in Intelligent Systems Designn (ISDA 2007).

[54] Göçken M, Özçalıcı M, Boru A, and Dosdoğru AT. Integrating metaheuristics and artificial neural networks for improved stock price prediction. *Expert Systems with Applications*. 2016, 44:320–331. https://doi.org/10.1016/j.eswa.2015.09.029. Available from: https://www.sciencedirect.com/science/article/pii/S0957417415006570.

[55] White H. Learning in artificial neural networks: A statistical perspective. *Neural computation*. 1989, 1(4):425–464. https://doi.org/10.1162/neco.1989.1.4.425. Available from: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.4.425?mobileUi=0.

[56] Burrell PR and Folarin BO. The impact of neural networks in finance. *Neural Computing & Applications*. 1997, 6(4):193–200.

[57] Garliauskas A. Neural network chaos and computational algorithms of forecast in finance. In: *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, volume 2, p. 638–643 vol.2, 1999. https://doi.org/10.1109/ICSMC.1999.825335. Available from: https://www.researchgate.net/publication/267791105_Neural_network_chaos_analysis/fulltext/54b432420cf26833efd0080f/Neural-network-chaos-analysis.pdf.

[58] Leung MT, Daouk H, and Chen AS. Forecasting stock indices: a comparison of classification and level estimation models. *International Journal of forecasting*. 2000, 16 (2):173–190. https://doi.org/10.1016/S0169-2070(99)00048-5. Available from: https://www.sciencedirect.com/science/article/pii/S0169207099000485.

[59] Chen AS, Mark TL, and Daouk H. Application of neural networks to an emerging financial market: forecasting and trading the taiwan stock index. *Computers Operations Research*. 2003, 30(6):901 – 923. ISSN 0305-0548. https://doi.org/10.1016/S0305-0548(02)00037-0. Available from: http://www.sciencedirect.com/science/article/pii/S0305054802000370. Operation Research in Emerging Economics.

[60] Martinez LC, da Hora DN, Palotti J, Meira W, and Pappa GL. From an artificial neural network to a stock market day-trading system: A case study on the bm&f bovespa. In: *2009 International Joint Conference on Neural Networks*, p. 2006–2013. IEEE, 2009. https://doi.org/10.1109/IJCNN.2009.5179050. Available from: https://ieeexplore.ieee.org/document/5179050.

[61] Thawornwong S and Enke D. The adaptive selection of financial and economic variables for use with artificial neural networks. *Neurocomputing*. 2004, 56:205–232. https://doi.org/10.1016/j.neucom.2003.05.001.

[62] Cao R, Leggio K, and Schniederjans M. A comparison between fama and french's model and artificial neural networks in predicting the chinese stock market. *Computers Operations Research*. 10 2005, 32:2499–2512. 10.1016/j.cor.2004.03.015.

Available from: https://www.sciencedirect.com/science/article/pii/S030505480400067X?via%3Dihub.

[63] Wu L and Shahidehpour M. A hybrid model for day-ahead price forecasting. *IEEE Transactions on Power Systems*. 2010, 25(3):1519–1530. https://doi.org/10.1109/TPWRS.2009.2039948. Available from: https://ieeexplore.ieee.org/document/5409501.

[64] Liang X, Zhang H, Xiao J, and Chen Y. Improving option price forecasts with neural networks and support vector regressions. *Neurocomputing*. 2009, 72(13-15):3055–3065. https://doi.org/10.1016/j.neucom.2009.03.015. Available from: https://www.sciencedirect.com/science/article/pii/S092523120900109X?via%3Dihub.

[65] Yu L, Chen H, Wang S, and Lai KK. Evolving least squares support vector machines for stock market trend mining. *IEEE Transactions on Evolutionary Computation*. 2009, 13(1):87–102. https://doi.org/10.1109/TEVC.2008.928176. Available from: https://ieeexplore.ieee.org/document/4632148.

[66] Leigh W, Purvis R, and Ragusa J. Forecasting the nyse composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: A case study in romantic decision support. *Decision Support Systems*. 03 2002, 32:361–377. 10.1016/S0167-9236(01)00121-X. Available from: https://www.sciencedirect.com/science/article/pii/S016792360100121X?via%3Dihub.

[67] Kim KJ and Han I. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*. 2000, 19(2):125–132. https://doi.org/10.1016/S0957-4174(00)00027-0. Available from: https://www.sciencedirect.com/science/article/pii/S0957417400000270?via%3Dihub.

[68] Phua P, Ming D, and Lin W. Neural network with genetically evolved algorithms for stocks prediction. *Asia Pacific Journal of Operational Research*. 05 2001, 18:103–107. Available from: https://www.researchgate.net/journal/0217-5959_Asia_Pacific_Journal_of_Operational_Research.

[69] Sezer OB, Gudelek MU, and Ozbayoglu AM. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*. 2020, 90:106181. https://doi.org/10.1016/j.asoc.2020.106181. Available from: https://www.sciencedirect.com/science/article/pii/S1568494620301216?via%3Dihub.

[70] Rustam Z and Saragih GS. Predicting bank financial failures using random forest. In: *2018 International Workshop on Big Data and Information Security (IWBIS)*, p. 81–86. IEEE, 2018. https://doi.org/10.1016/j.cose.2015.09.005. Available from: https://www.sciencedirect.com/science/article/pii/S0167404815001261?via%3Dihub.

[71] West J and Bhattacharya M. Intelligent financial fraud detection: a comprehensive review. *Computers & security*. 2016, 57:47–66. https://doi.org/10.1016/j.cose.2015.09.005.

Available from: https://www.sciencedirect.com/science/article/pii/S0167404815001261?via%3Dihub.

[72] Kumar M and Thenmozhi M. Forecasting stock index returns using arima-svm, arima-ann, and arima-random forest hybrid models. *International Journal of Banking, Accounting and Finance*. 2014, 5(3):284–308. https://doi.org/10.1504/IJBAAF.2014.064307. Available from: https://www.inderscienceonline.com/doi/abs/10.1504/IJBAAF.2014.064307.

[73] Abdelmaksoud A. Dissertation: Forecasting gspc using random forest, long short-term memory neural network, support vector classifier, and feed forward neural network. https://github.com/aabdelmak/Dissertation/blob/master/Dissertation_saved.ipynb, 2020.

[74] Nelson DMQ, Pereira ACM, and de Oliveira RA. Stock market's price movement prediction with lstm neural networks. In: *2017 International Joint Conference on Neural Networks (IJCNN)*, p. 1419–1426, 2017. Available from: "https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7966019".