https://doi.org/10.51483/IJAIML.4.2.2024.106-133 ISSN: 2789-2557

SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

A Comparison of Standard Statistical, Machine Learning and Deep Learning Methods in Forecasting the Time Series

Krishnandu Ghosh[1*]

[1]Indira Gandhi Institute of Development Research, Gen A.K. Vaidya Marg, Goregaon(E), Mumbai 400065, India. E-mail: krishnandu@igidr.ac.in

Article Info

Volume 4, Issue 2, July 2024
Received : 14 February 2024
Accepted : 02 June 2024
Published : 05 July 2024
doi: 10.51483/IJAIML.4.2.2024.106-133

Abstract

Macroeconomic indicator forecasting is a difficult task and the macroeconomy's complex operations and dynamic nature make it even more difficult. Machine Learning and Deep Learning methodologies have been investigated as alternatives to traditional forecasting methods because of recent developments in computing power and the emergence of data. How the Machine Learning and Deep Learning paradigms apply to a variety of Macro datasets have been examined in this research paper. Few Machine Learning and Deep Learning algorithms have been trained and their forecasting accuracy has been compared with that of traditional statistical method ARIMA.

Keywords: *Time series, Forecasting, Machine learning, Deep learning, Statistical methods*

1. Introduction

Traditional macroeconomic forecasting methods include structural forecasting and non-structural forecasting. Economic theory directs the structural approach to model design, whereas the non-structural method focuses on utilizing the unique characteristics of the underlying data without explicitly relying on any economic theory. For short-term unconditional forecasting, non-structural models are advantageous because they prioritise predictive accuracy over causal inference. The multivariate vector autoregression (VAR) and the univariate autoregressive integrated moving average (ARIMA) are two popular non-structural time-series models. Such time-series models rely heavily on parameter linearity. Linear models were unable to predict macroeconomic business cycles, periods of excessive volatility, and regime shifts, which led to the rise in popularity of non-linear models.

Machine Learning (ML) techniques have lately been put out in the forecasting literature as an alternative to statistical models. The application of ML approaches, particularly Neural Networks, for time-series predictions, methodological breakthroughs, and accuracy improvements is the subject of a lot of research due to advancements in processing and the accessibility of high-frequency data. Both statistical and machine learning

* *Corresponding author: Krishnandu Ghosh, Indira Gandhi Institute of Development Research, Gen A.K. Vaidya Marg, Goregaon(E), Mumbai 400065, India. E-mail: krishnandu@igidr.ac.in*

models seek to increase prediction accuracy by minimizing a loss function, but they differ in the methods they use to do so; the former uses linear processes, while the latter uses non-linear techniques. On the other hand, it would be incorrect to assume that ML models are always superior to statistical models. The goal indicator, underlying data generation process (DGP), prediction horizon, and data quality will probably all be important factors in this empirical inquiry, which needs to be thoroughly examined.

The main goal of this paper is to compare statistical method ARIMA with ML methods like Random Forests (RF) and Support Vector Regression (SVR) and Deep Learning (DL) Methods like Artificial Neural Network (ANN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to forecast Time Series Data.

2. Application of Machine Learning in Economics

It is challenging to predict stock prices, and most academic circles have always believed that stock values are unpredictable, in line with the efficient market hypothesis (Malkiel, 2003). Recent research has identified momentum, economic, and psychological behaviour factors as viable sources for stock price forecasting (Gray and Vogel, 2016; Lo *et al.*, 2000; Christoffersen and Diebold, 2006; Moskowitz *et al.*, 2012). Furthermore, recent research on stock price predictability suggests that forecasting stock price direction is more successful than forecasting actual stock prices (Basak *et al.*, 2019; Nyberg, 2011; Nyberg and Pönkä, 2016; Ballings *et al.*, 2015; Lohrmann and Luukka, 2019).

A tree classifier combined with recurrent neural networks, according to Mallqui and Fernandes (2019), is better at forecasting Bitcoin price direction than SVM. Mokoaleli-Mokoteli *et al.* (2019) compare the performance of boosted, bagged, RUS-boosted, subspace disc, and subspace k-nearest neighbour (KNN) ML ensemble approaches in forecasting the direction of the Johannesburg Stock Exchange's stock price. Boosted approaches beat KNN, logistic regression, and SVM, according to the researchers. Nti *et al.* (2020) claim that RFs are underutilised in studies that concentrate on stock price forecasting. To anticipate stock prices, Weng *et al.* (2018) combine social media-type data such as web page views, sentiment in financial news and search patterns with techniques in machine learning (boosted regression tree, RFs, ANN, SVM).

Support vectors for local approximation have been used to apply this idea to regression, enabling the estimation of nonlinear regression through SVRs (SVRs). SVRs have been widely used in finance and other fields for forecasting purposes (Tay and Cao, 2001, 2002; Kim, 2003; Pai and Lin, 2005; Huang *et al.*, 2005; Chen *et al.*, 2006).

3. Statistical Based Model

3.1. ARIMA

In statistics and econometrics, an autoregressive integrated moving average (ARIMA) model is a generalisation of an autoregressive moving average (ARMA) model, particularly in time series analysis. Both models are used to analyze time series data, either to better understand it or to predict future points in the series (forecasting). When data show evidence of mean function non-stationarity (but not variance/autocovariance), an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate mean function non-stationarity (i.e., the trend).

The AR component of ARIMA denotes that the evolving variable of interest is regressed on the values of its own lag (i.e., prior). The MA component indicates that the regression error is a linear combination of error terms that occurred concurrently and at different points in the past. The letter I (for "integrated") indicates that the data values have had their current values replaced with the difference between their current and previous values (and this differencing process may have been performed more than once). Each of these features is intended to make the model as close to the data as possible.

ARIMA (p, d, q) denotes non-seasonal ARIMA models, where p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data has had past values subtracted), and q is the order of the moving-average model.

Given time series data X_t where t is an integer index and the X_t are real numbers, an ARMA (p', q) model is given by

$$X_t - \alpha_1 X_{t-1} - \ldots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}$$

or equivalently by

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

where L is the lag operator, the α_t are the parameters of the autoregressive part of the model, the θ_t are the parameters of the moving average part and the ε_t are error terms. In general, the error terms are assumed to be independent, identically distributed variables sampled from a normal distribution with a zero mean.

Assume now that the polynomial $\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right)$ has a unit root (a factor (1 - L)) of multiplicity d. Then it can be rewritten as

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) = \left(1 - \sum_{i=1}^{p'-d} \varphi_i L^i\right) (1 - L)^d$$

This polynomial factorization property is expressed by an ARIMA (p, d, q) process, which has $p = p' - d$ and is given by

$$\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right)(1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

As a result, it can be thought of as a special case of an ARMA (p+d, q) process with autoregressive polynomial with 'd' unit roots.

4. Machine Learning Based Model

4.1. Random Forest

Random forest is a frequently used Supervised Machine Learning Algorithm for classification and regression problems. It builds decision trees from different samples, using the average for regression and the majority vote for classification. One of the most important characteristics of the Random Forest Algorithm is that it can handle data sets with both continuous variables (as in regression) and categorical variables (as in classification). It outperforms other algorithms in classification problems.

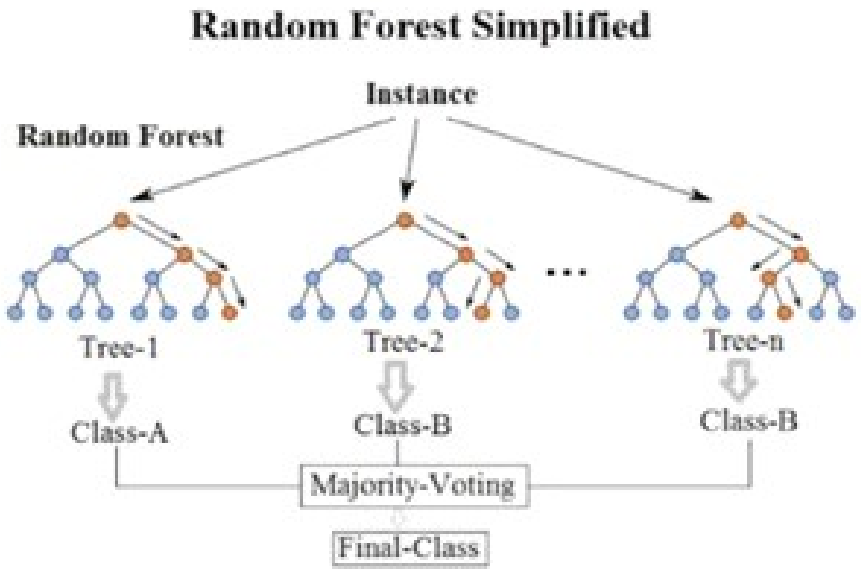

Figure 1: Random Forest

We must first comprehend the ensemble technique to comprehend how the Random Forest functions. The combination of various models is referred to as "ensemble." As a result, multiple models are employed to create predictions as opposed to only one.

Ensemble uses two different types of techniques:

Bagging: A fresh training subset is created using sample training data and replacement, and the outcome is decided by a majority vote. Take for instance, Random Forest.

Boosting: It converts weak learners into strong ones by building the most accurate sequential models. ADA BOOST and XG BOOST are two examples.

4.1.1. The Random Forest Algorithm's Steps are as Follows

Step 1: In Random Forest, *n* random records are chosen at random from a data set with *k* records.

Step 2: For each sample, a unique decision tree is built.

Step 3: Each decision tree will produce a result.

Step 4: The ultimate result for classification and regression is based on majority voting or averaging.

4.1.2. Important Random Forest Features

Diversity: Not all characteristics, variables, or features are considered when making a specific tree because every tree is different.

Immune to the Curse of Dimensionality: The feature space is reduced because each tree does not consider all the features (Figure 1).

Parallelization: Each tree is built independently from various data and attributes. This means that we can fully utilize the CPU to construct random forests.

Train-Test Split: We do not need to separate the data for train and test in a random forest because the decision tree will always miss 20% of the data.

Stability: The result is stable because it is based on majority voting/averaging.

4.1.3. Hyperparameters of the Random Forest Regression

In random forests, hyperparameters are used to either improve model performance and predictive power or to make the model faster.

4.1.4. The Prediction Power is Increased by Using the Following Hyperparameters

N Estimators: The number of trees the algorithm builds prior to averaging the predictions

Max Features: Considers splitting a node based on the maximum number of features

Mini Sample Leaf: Decides how many leaves are necessary to separate an internal node in the fewest possible number.

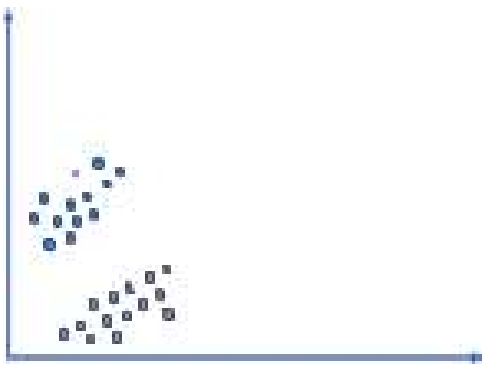

Figure 2: Showing Two Label Classes

4.2. Support Vector Machine

Let us define Support Vector Machine (SVM) in layman's terms. Assume we have the plot shown in the image below, which has two label classes (Figure 2).

How to you decide where to draw the dividing line? This could have been thought of by someone.

The classes are effectively divided by the line. SVM essentially performs simple class separation in this manner. What if the data were in this format? (Figure 3)

There is no simple line separating these two classes in this case. As a result, we will broaden our dimension and add a new dimension along the z-axis. We can now distinguish between these two classes.

This line maps to the circular border as depicted in this image when it is transformed back to its native plane (Figure 4).

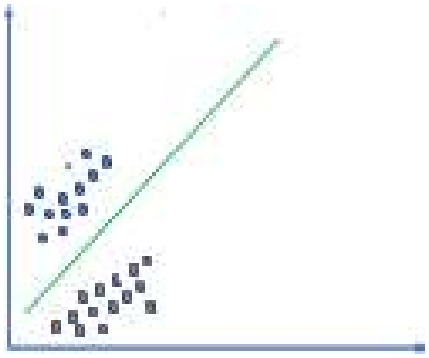

Figure 3: Line Dividing the Two Label Classes

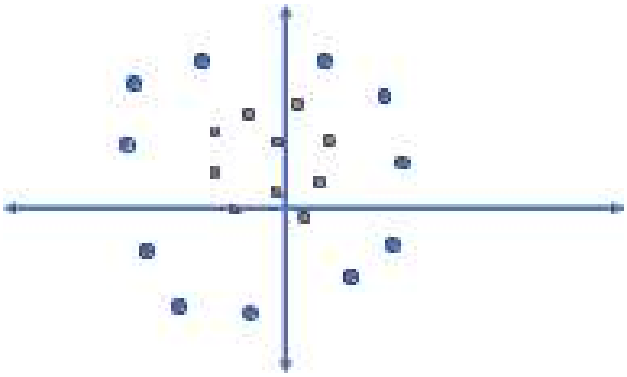

Figure 4: Two Label Classes in a Zig Zag Manner

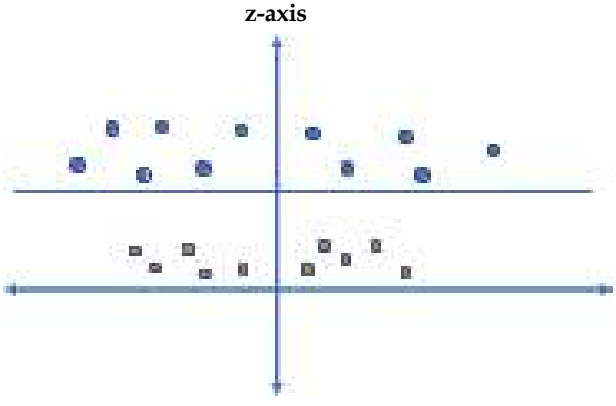

Figure 5: Adding a New Dimension along the Z-Axis

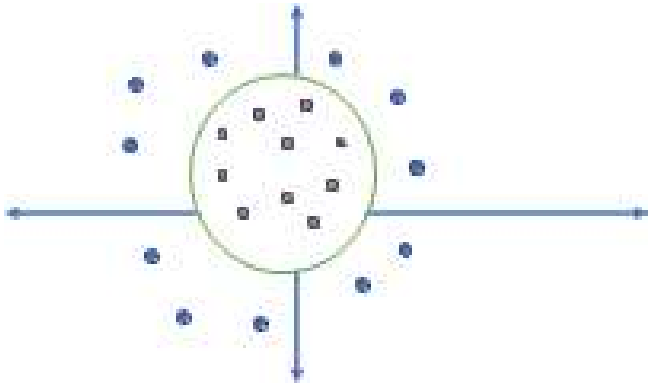

Figure 6: Mapping to the Circular Boundary

This is precisely what SVM accomplishes! It seeks a line/hyperplane (in multidimensional space) that divides these two classes. Depending on the classes to predict, the new point is then classified based on whether it is on the positive or negative side of the hyperplane.

4.2.1. Hyperparameters of the Support Vector Machine (SVM) Algorithm

Kernel: A kernel assists us in locating a hyperplane in higher dimensional space while minimizing computational cost. Typically, as the dimension of the data increases, so does the computational cost. When we cannot locate a separating hyperplane in a particular dimension and must travel to a higher dimension, this dimension increase is required.

Hyperplane: In SVM, this is essentially a dividing line between two data classes. This, however, is the line that will be used to predict the continuous output in Support Vector Regression.

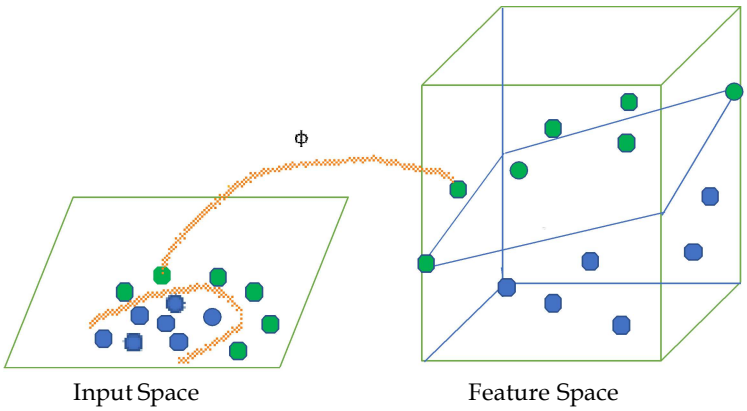

Figure 7: Mapping in a Higher Dimensional Space

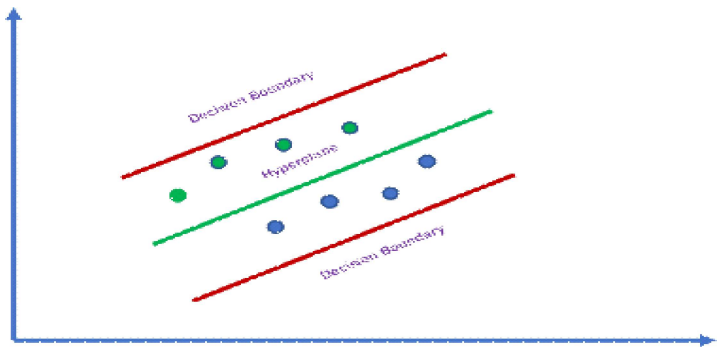

Figure 8: Support Vector Regression

Decision Boundary: For ease of understanding, imagine a decision boundary as a demarcation line with positive and negative examples on either side. Both positive and negative examples can be found on this line. The Support Vector Regression algorithm will also employ the same SVM principle.

4.2.2. Support Vector Regression

Support Vector Regression (SVR) is based on the same principle as SVM, but it is applied to regression problems.

Based on a training sample, the problem of regression is to find a function that approximates mapping from an input domain to real numbers.

Consider the green line to be the hyperplane and the two red lines to be the decision boundaries. When we move forward with SVR, our goal is to basically consider the points that are within the decision boundary line. The hyperplane with the greatest number of points is our best fit line.

The first thing we will learn is what the decision boundary is (the red line shown above). At any distance from the hyperplane, let us say 'a', consider these lines to be. We therefore draw these lines from the hyperplane at distances 'a' and '–a'.

A support-vector machine creates a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like detecting outliers. Intuitively, the hyperplane with the greatest distance to the nearest training-data point of any class (so-called functional margin) achieves a decent separation, because the larger the margin, the lower the classifier's generalisation error.

4.2.3. Linear SVM

We are given a training dataset of n points in the form $(x_1, y_1), \ldots, (x_n, y_n)$ where the y_i are either 1 or -1, indicating which class the point x_i belongs to. A real vector of p dimensions exists for each x_i. We are looking for the "maximum - margin hypothesis," which divides the group of points xi for which $y_i = 1$ from the group of points x_i for which $y_i = -1$, and is specified so that the distance between the hyperplane and the nearest point x_i from either group is maximized.

Any hyperplane can be expressed as the collection of points obeying the equation $w^T x - b = 0$, where w is the hyperplane's normal vector. With the exception of w not always being a unit vector, this is quite similar to Hesse Normal Form. The parameter $\frac{b}{\|w\|}$ determines the offset of the hyperplane from the origin along the normal vector w.

4.2.4. Hard Margin

If the training data is linearly separable, we can choose two parallel hyperplanes to separate the two classes of data with the greatest feasible distance between them. The "margin" is the region bounded by these two

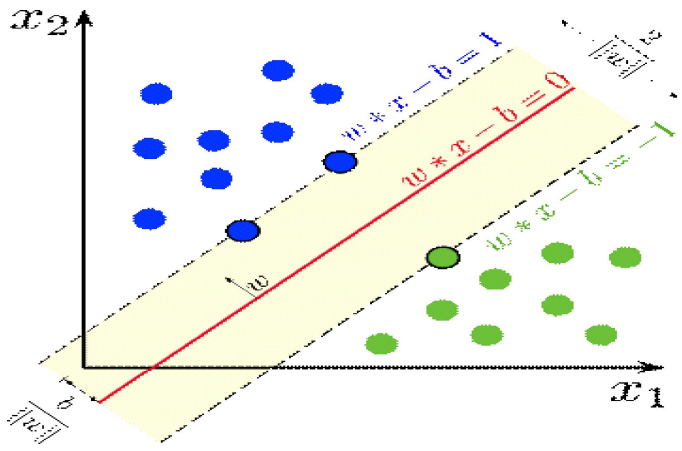

Figure 9: Hard Margin

hyperplanes, and the maximum margin hyperplane is the hyperplane midway between them. These hyperplanes can be defined by the equations with a normalized or standardized dataset as follows.

$w^T x - b = 1$ (anything on or above this boundary is of one class, with label 1) and

$w^T x - b = -1$ (anything on or below this boundary is of one class, with label -1)

The geometric distance between these two hyperplanes is $\frac{2}{\|w\|}$, thus we want to minimize $\|w\|$ to maximize the distance between the planes. The distance is calculated using the equation for distance from a point to a plane. We also need to keep data points out of the margin, so we add the following restriction.

$y_i(w^T x_i - b) \geq 1$ for all $1 \leq i \leq n$. The optimization problem can now be stated as follows - Minimize $\|w\|$ subject to the condition $y_i(w^T x_i - b) \geq 1$ for all $1 \leq i \leq n$

4.2.5. Soft Margin

The hinge loss function is useful for extending SVM to circumstances when the data are not linearly separable. $\max(0, 1 - y_i(w^T x_i - b))$. Note that y_i is the i^{th} target and $(w^T x_i - b)$ is the i^{th} output.

Then the goal of the optimization is to minimize

$$\lambda \|w\|^2 + \frac{\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(w^T x_i - b\right)\right)}{n}$$

where the trade-off between raising the margin size and guaranteeing that the x_i lie on the correct side of the margin is determined by the parameter $\lambda > 0$. If the input data are linearly classifiable, it will behave similarly to the hard-margin SVM for sufficiently small values of λ, but it will still learn whether a classification rule is viable or not. 'C' is another name for this parameter.

5. Deep Learning Based Model

5.1. Artificial Neural Network

Artificial Neural Networks (ANN) are brain-inspired algorithms that are used to foresee problems and model complex patterns. The idea of biological neural networks in the human brain gave rise to the Artificial Neural Network (ANN), a deep learning technique. An effort to simulate how the human brain functions led to the creation of ANN.

5.1.1. Artificial Neural Networks and its Components

Neural Networks are a type of computational learning system that uses a network of functions to understand and translate a data input in one form into a desired output, typically in another. Human biology and how neurons in the human brain collaborate to understand input from human senses inspired the concept of an artificial neural network. In a nutshell, Neural Networks are a collection of algorithms that attempt to recognize patterns, relationships, and information in data using a process inspired by and working similarly to the human brain/biology.

5.1.2. Different Layers of a Neural Network

Input Layer: This is the gateway for the inputs that we provide.

Hidden Layers: This is the layer where complex computations take place. The more hidden layers you have in your model, the more complex it will be. This is similar to a neural network's black box, where complicated relationships in the data are learned by the model.

Output Layer: Although the output layer has just one node in the diagram above, this is not always the case in each model of a neural network. How many nodes are in the output layer is based on the issue that we have selected. The number of nodes in the output layer will match the number of classes to predict if the model is a classification one.

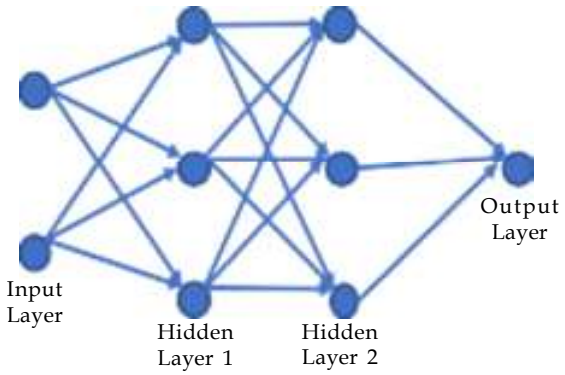

Figure 10: Neural Network

5.1.3. Forward Propagation

Through the hidden layers, we are pushing the input to the output layer during forward propagation. Another intriguing finding is that the hidden layer merely contains a few neurons which fire throughout the entire process. However, there is a considerable risk that if the model is poorly trained it will make an incorrect prediction, introducing error.

Perceptrons are the simplest neural network, consisting of n number of inputs, only one neuron, and one output, where n is the number of features in our dataset. They were invented by Frank Rosenblatt in 1958. Three sections are used to explain how forward propagation works in perceptrons.

Step 1: For each input, multiply the input value x_i with the weights w_i for each input and add the results. Weights determine how much influence a certain input has on a neuron's output by representing the strength of the link between neurons. If the weight w_1 is greater than the weight w_2, the input x_1 will have a greater influence on the output than the weight w_2.

$$\sum = x_1 \cdot w_1 + \ldots + x_n \cdot w_n$$

$x = (x_1, x_2, \ldots, x_n)$ and $w = (w_1, w_2, \ldots, w_n)$ are the row vectors for the inputs and weights, respectively, and their dot product is given by

$$x \cdot w = x_1 \cdot w_1 + \ldots + x_n \cdot w_n$$

Step 2: Let us name this z by adding bias b to the sum of multiplied values. In most circumstances, bias—also known as offset—is required to move the entire activation function to the left or right in order to generate the required output values.

$$z = x \cdot w + b$$

Step 3: To a non-linear activation function, pass the value of z. To provide nonlinearity to the neurons' output, activation functions are used; otherwise, the neural network would function linearly. They also significantly affect how quickly the neural network learns. Perceptron's activation function is a binary step function. But we will use the sigmoid, often known as the logistic function, for our activation function.

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

where σ represents the sigmoid activation function, and $\hat{y}$ the anticipated value is the output obtained after forward propagation.

5.1.4. Algorithm for Learning

The backpropagation and optimization phases make up the learning algorithm.

5.1.5. Backpropagation

The procedure for computing the gradient of the loss function with respect to the weights is known as

backpropagation, which is short for backward propagation of error. The word is, however, frequently misused to refer to the complete learning algorithm. In the following two steps, we will go through how a perceptron performs backpropagation.

Step 1: A loss function is used to determine how far we are from our target solution. For regression issues, the loss function is usually mean squared error, and for classification problems, cross entropy. Consider a regression issue with a mean squared error loss function, which squares the difference between the actual y_i and the anticipated value $\hat{y}_i$.

$$MSE_i = \left(y_i - \hat{y}_i\right)^2$$

The cost function *C* is the average of the loss functions derived for the full training dataset.

$$C = MSE = \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}{n}$$

Step 2: We need to know how the cost function varies in respect to weights and bias in order to identify the best weights and bias for our perceptron. Gradients (rate of change) i.e., how one quantity varies in proportion to another are used to do this. In our scenario, we need to determine the cost function's gradient in relation to the weights and bias.

Let us use partial derivation to calculate the gradient of the cost function *C* with regard to the weight w_i. We will utilize the chain rule because the cost function is not directly related to the weight w_i.

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

The next step is to find the subsequent three gradients

$$\frac{\partial C}{\partial \hat{v}} = ?; \frac{\partial \hat{y}}{\partial z} = ?; \frac{\partial z}{\partial w_i} = ?$$

Let us start by looking at the *C* gradient of the cost function in relation to the predicted value $\hat{y}$.

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_1\right)}{n}}{\partial \hat{y}} = 2 \times \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_1\right)}{n}$$

$$\Rightarrow \frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times sum\left(y - \hat{y}\right)$$

Now $\frac{\partial \hat{y}}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{\partial\left(\frac{1}{1+e^{-z}}\right)}{\partial z} = \sigma(z) \times \left(1 - \sigma(z)\right)$

and $\frac{\partial z}{\partial w_i} = \frac{\partial \sum_{i=1}^{n}\left(x_i \cdot w_i + b\right)}{\partial w_i} = w_i$

Therefore, we get, $\frac{\partial C}{w_i} = \frac{2}{n} \times sum\left(y - \hat{y}\right) \times \sigma(z) \times \left(1 - \sigma(z)\right) \times x^i$

What about Bias? — Bias is theoretically considered to have an input of constant value 1.

Hence, $\frac{\partial C}{\partial b} = \frac{2}{n} \times sum\left(y - \hat{y}\right) \times \sigma(z) \times \left(1 - \sigma(z)\right)$

5.1.6. Optimization

It involves choosing the best element from a list of alternatives, in this case the best weights and bias of the perceptron. Let us utilize gradient descent as our optimization strategy, which modifies the weights and bias in proportion to the inverse of the gradient of the cost function with regard to the weights or bias. The amount that the weights and bias are changed is determined by a hyperparameter termed as learning rate "α".

Backpropagation and gradient descent are performed until convergence, and the weights and bias are updated as follows.

$$w_i = w_i - \left(\alpha \times \frac{\partial C}{\partial w_i} \right)$$

$$b = b - \left(\alpha \times \frac{\partial C}{\partial b} \right)$$

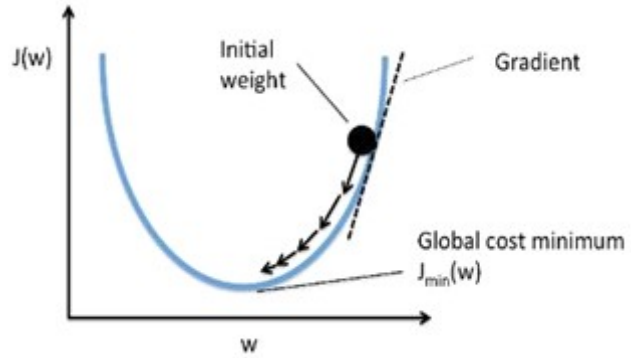

Figure 11: Gradient Descent

This is a critical step during the training process in which the Neural Network improves by learning from its errors and modifying its characteristics to minimize loss. The Neural Network updates its weights using an optimization method like gradient descent to find a specific point where the loss is minimal.

5.1.7. Activation Functions

In neural networks, activation functions are crucial. Both the output layer and the outputs of hidden layers frequently employ them.

5.1.8. Commonly Used Activation Functions

Rectified Linear Activation Function (ReLu) has served as a sort of default activation function in numerous Neural Network models. The outputs of a model created for classification-type tasks mostly use the Softmax and Sigmoid activation functions (Logistic).

Table 1: Activation Functions

Name	Equation
Identity	$f(x) = x$
Binary Step	$f(x) = \begin{cases} 0; & x < 0 \\ 1; & x \geq 0 \end{cases}$
Logistic (Soft Step)	$f(x) = \frac{e^x}{1+e^x}$
Rectified Linear Unit (ReLU)	$f(x) = \begin{cases} 0; & x < 0 \\ x; & x \geq 0 \end{cases}$
Soft Plus	$f(x) = \log_e(1 + e^x)$

For binary classification, the probability of classes is calculated using the Sigmoid function, whereas for multi-class classification issues, the probability is calculated using the Softmax activation function (Table 1).

5.1.9. Some Simple Optimization Functions Stochastic Gradient Descent (SGD)

An alternative to loading the full dataset is stochastic gradient descent (SGD), which computes the derivative of the parameters by only considering one particular training sample at a time. The SGD's time complexity and convergent time to the global minima were, however, much higher than those of the conventional gradient descent technique.

5.1.10. Mini Batch Stochastic Gradient Descent (MB-SGD)

A small deviation from the typical SGD is the Mini Batch SGD. Instead of taking a sample one at a time, the parameters are modified after collecting a group of samples.

5.2. Convolutional Neural Network Introduction

A Convolutional Neural Network (CNN) is commonly used in deep learning to analyze visual imagery, image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces and financial time series.

The term "convolutional neural network" refers to the network's use of a mathematical operation known as convolution. Convolutional neural networks are a subset of neural networks that use convolution rather than general matrix multiplication in at least one of their layers.

5.2.1. Convolutional Neural Networks and its Components

The importance of each algorithm's layers of neurons is that each layer concentrates each pattern and its significance, such as first observing simple patterns like lines and curves, then some complex patterns like texts, objects, and faces, and finally more complex patterns present in the image or whatever inputs we are analyzing.

This process is similar to that of a human in that when we are randomly visualizing or seeing something, our brain analyses any input in the same way, but in a matter of seconds.

So, in order to achieve this CNN algorithm, the following steps must be taken—Convolution Layer, Activation Function (ReLu layer), Pooling, Flattening and Full Connection.

5.2.2. Convolution Layer

It is the CNN algorithm's most important portion or segment. Convolution is denoted mathematically as $(f*g)(t)=\int_{-\infty}^{\infty} f(\tau)\,g(t-\tau)\,d\tau$

Convolution can be defined as the integration of two simple functions (f and g). We have two terms, i.e., input/dataset in matrix format and feature detector or kernel or filters (it will be a 3*3 or 5*5 or 7*7 matrix, depending on the architecture). This matrix is smaller than the input image matrix, but we can learn more from this kernel part alone.

So, in simple terms, a convolution block is used to extract features from a given dataset by avoiding other unimportant features or dimming or avoiding useless information or features.

Filters play an important role in transforming data (numbers/pixels), and the resulting output is referred to as activation maps, which are regions where specific kernel features can be detected from the input data.

The value of the weight and bias must be discovered in the subsequent optimization phase in order to generate the best suited output feature map. The output feature for one filter is referred to as a feature map.

To begin, all of the variables are classified as real number fields (R), and we define some of the key terms below. Filter (or kernel) is a three-dimensional matrix of size K × K × F, where K × K denotes a single channel aspect and F is the number of channels. Because the number of channels in a filter is equal to the number of channels in the input image, the channel is also known as the image depth.

Table 2: Convolution

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	K_{11}	K_{12}	K_{13}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	K_{21}	K_{22}	K_{23}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	K_{31}	K_{32}	K_{33}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}			
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}			

The number of filters, which also refers to the number of output feature maps from the convolutional process, is the depth of the filters. For example, given the 227 × 227 × 3 input image, the first convolutional layer of the AlexNet's first convolutional layer has a filter of size 11 × 11 × 3 with a depth of 96. It is worth noting that the filter's number of channels is the same as the number of channels in the input image. The convolutional layer then generates 96 feature maps (55 × 55 × 96).

The convolutional operation is a mixture of four summations, which is worth noticing. The first two sums over one channel of a filter K×K, the third adds over the channel (F), and the last one sums over the depth of the filters (D), from inner to outer. In order to see how the operation applies to an image, imagine F = 1, D = 1, and start with a 5 × 5 image and a 3 × 3 filter (or kernel). Convolution is calculated by sliding the filter across the image from the top left corner. The first value of the output matrices is then calculated by adding the product of the pixel value and the filter value.

In other words, we may mathematically define the operation for a single channel as

$$O(i,\ j) = \sum_{k=1}^{K} \sum_{l=1}^{K} I(i+k-1,\ j+l-1) K(k,\ l)$$

where $K \times K$ is the size of the filter, O denotes output, and $O(i, j)$ is the value of the i^{th} row and j^{th} column of the output matrix. Furthermore, i is running from 1 to $W - K + 1$ and j is running from 1 to $H - K + 1$ where $W \times H$ is the size of the image.

5.2.3. Activation Function (ReLU)

So, once the convolution process for our input dataset is complete, the activation function is the next critical step in the CNN algorithm.

The activation function, in contrast to a neuron-based model seen in our brains, is ultimately in charge of selecting which neuron to fire next. It takes the output signal from the preceding cell and transforms it so that the next cell can utilize it as input. We perform a non-linear transformation on inputs before sending them to the next layer or neuron.

A nonlinear activation function called the Rectified Linear Unit (ReLU) behaves mathematically like a biological neuron. Although standard activation functions like sigmoid or hyperbolic tangent (tan*h*) fulfil the same goal, it has been discovered that a CNN with ReLU as an activation function trains much quicker than a network with hyperbolic tangent as its activation function. Furthermore, ReLU is a fairly basic function, with the formula $f(x) = \max(0, x)$, where the function is linear for $x > 0$ and 0 for $x < 0$. In other words, the function replaces all negative values with 0 while leaving the positive value unchanged. It is worth noting that, like the pooling layer, there are no parameters that need to be learned.

5.2.4. Pooling

The pooling process reduces the dimensionality of our matrix. The primary benefit of pooling is that it reduces overfitting and computation in our dataset.

It is classified into two types - Max-Pooling and Avg-Pooling.

The pooling procedure, also known as the sub-sampling layer, applies a down sampled operation to the input feature map, also known as dimension reduction. It is worth noting that the pooling process does not necessitate the training of additional parameters. In general, there are two types of pooling operations: average pooling and maximum pooling. The pooling layer, like the convolutional layer, has hyperparameters such as filter size (R), stride (S), and padding (P).

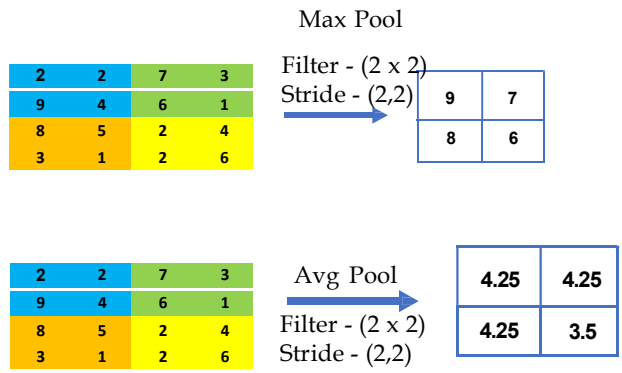

Figure 12: Pooling

One of the most common pooling operations is max pooling, which is just a max function that filters the input map and extracts the maximum value from the filtered zone. $O(i,\ j) = \max_{i \le l \le i+R-1,\ j \le k \le j+R-1} \{I(l,\ k)\}$ is the definition of the operation, where $O(i, j)$ is the value of the i^{th} row and j^{th} column of the output matrix.

5.2.5. Flattening

The next layer after the pooling process is flattening, in which we will convert our matrix into a single column, essentially taking row by row numbers and putting them into a single column.

Flattening is the process of converting data into a one-dimensional array for input into the next layer. The convolutional layer output is flattened to produce a single long feature vector. It is also linked to the final classification model, which is referred to as a fully connected layer.

In other words, we combine all of the pixel data into a single line and connect it to the final layer. And once more. What is the purpose of the final layer? The categorization of 'the cats and dogs', or normal and abnormal, good or bad, or any categorization.

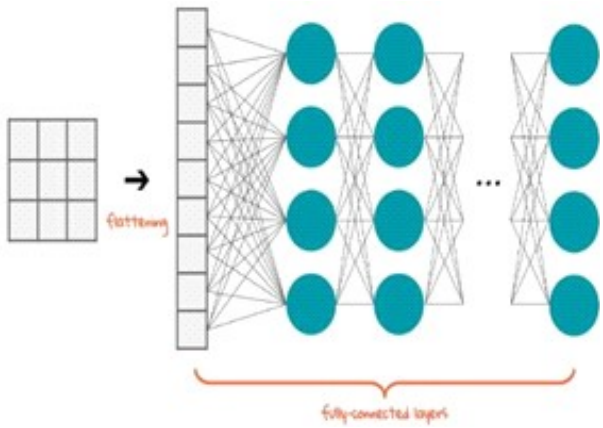

Figure 13: Flattening

5.2.6. Fully Connected Layer

The final layer, in which each pixel or number/point is treated as a separate neuron, similar to a Neural Network. The number of neurons in the final fully-connected layer will be equal to the anticipated number of classes.

Now that we have converted our input image into a format suitable for our multi-level fully connected architecture, we will flatten it into a single column vector. The flattened output is fed into a feed-forward neural network, and backpropagation is used for each training iteration. The model can distinguish and classify dominating and certain low-level features in images over a series of epochs. This layer predicts a suitable label by assigning random weights to the inputs. Aside from these blocks, some special blocks are used for CNN.

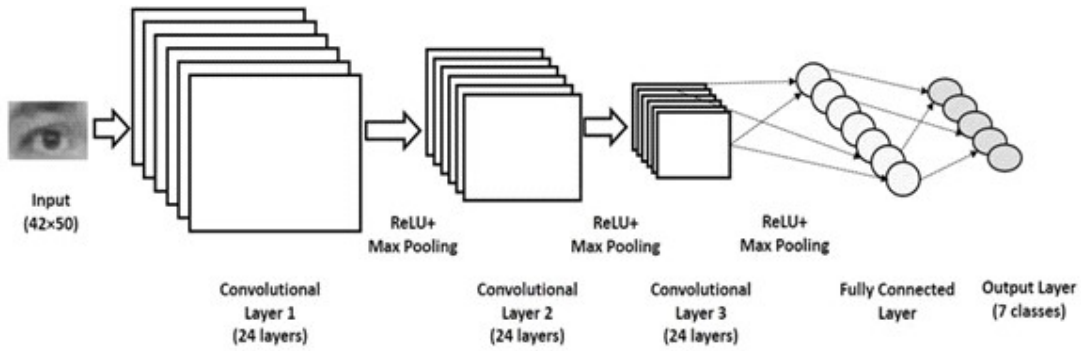

Figure 14: Fully Connected Layer

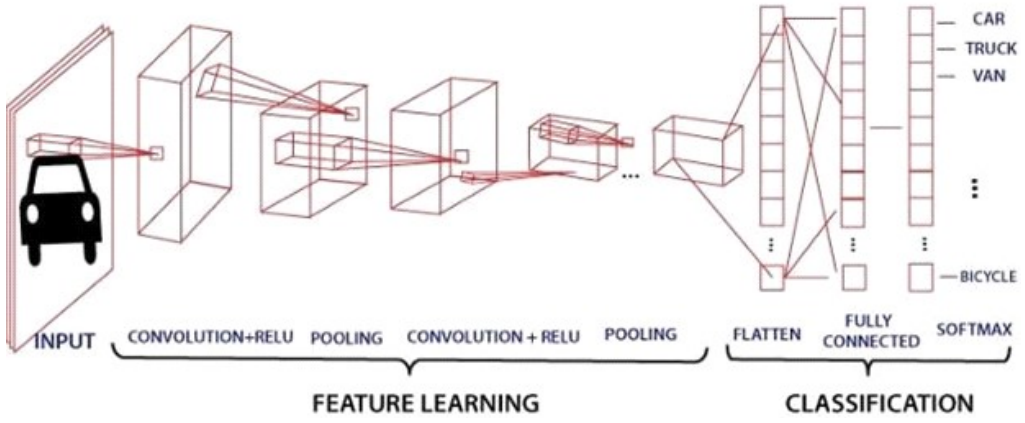

Figure 15: Final Layer of the CNN Model

The final layer of the CNN model contains the classification label results and assigns a class to the input dataset.

5.3. Recurrent Neural Network

An artificial neural network called a recurrent neural network (RNN) is made to process time series data or data that comprises sequences. Only unrelated data points should be used with conventional feed forward neural networks. However, we must adjust the neural network to take into account dependencies if we have data in a sequence where one data point depends on the preceding data point. In order to produce the following output in a series, RNNs contain the idea of "memory", which enables them to store the states or information of prior inputs.

Let us have a look at the historical Simple Recurrent Networks proposed by Elman (1990) and Jordan (1990). Elman considers a recurrent network that is an MLP with one unit layer looped back on itself. The

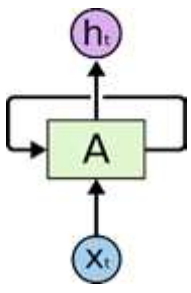

Figure 16: Recurrent Neural Networks have Loops

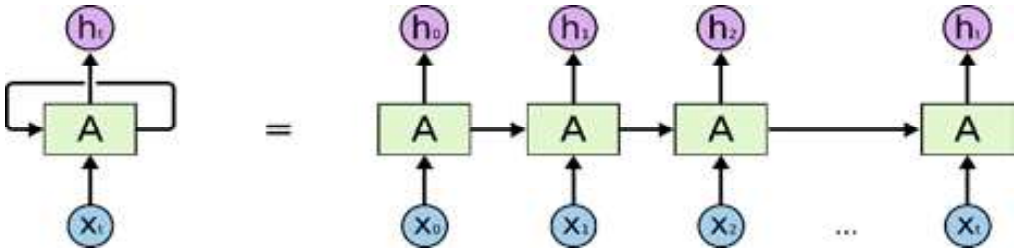

Figure 17: An Unrolled Recurrent Neural Network

model writes for the k^{th} component of the output if we indicate by $x(t)$ the input at time t, $\hat{y}(t)$ the output at time t, and $\hat{z}(t)$ the hidden layer at time t.

$$\hat{y}^{(k)}(t) = \sum_{i=1}^{I} w_i^{(k)} \hat{z}_i(t) + b^{(k)}$$

$$\hat{z}_i(t) = \sigma\left(\sum_{j=1}^{J} w_{i,j} x_j(t) \sum_{l=1}^{I} \widehat{w_{i,l}} \hat{z}_l(t-1) + b_i\right)$$

where σ is an activation function. Context units are the neurons in the hidden layer that are looped to themselves. In Jordan's model, $\hat{z}_l(t-1)$ is replaced by $\hat{y}_l(t-1)$ in the last equation. The output neurons in this scenario are the context units. In linguistic analysis, these models have been introduced. Natural language processing makes extensive use of them. Despite this, the most basic version of recurrent neural networks struggles to learn long-term dependencies. To address this issue, new architectures have been introduced.

Recurrent neural networks are intricately tied to sequences and lists, as seen by their chain-like character. They are the most natural neural network architecture to employ for such data.

RNNs have had remarkable success in the previous several years when applied to a number of tasks, including speech recognition, language modelling, translation, and image captioning.

These achievements would not have been possible without the use of "LSTMs," a very particular type of recurrent neural network that, for many tasks, outperforms the standard version by a wide margin. Almost all of the most fascinating recurrent neural network results are achieved with them.

5.3.1. The Activation Function

In the recurrent neural network, we can use any activation function we want. Common choices are Sigmoid function, Tanh function and ReLu function.

5.3.2. Training A Recurrent Neural Network

An artificial neural network's backpropagation algorithm is modified to include time unfolding in order to train the network's weights. This algorithm is known as the back propagation in time algorithm, or BPTT algorithm for short. It is based on computing the gradient vector.

5.4. Long Short-Term Memory (LSTM)

Long Short-Term Memory networks, or "LSTMs," are a type of RNN that can learn long-term dependencies. Hochreiter and Schmidhuber (1997) introduced them, and numerous individuals developed and popularized

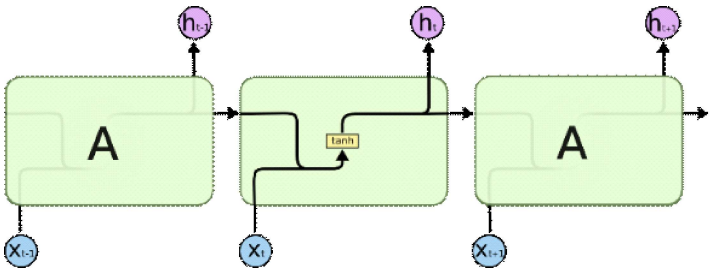

Figure 18: The Repeating Module in a Standard RNN Contains a Single Layer

them in subsequent work. They are currently frequently utilized and function exceptionally effectively on a wide range of situations.

LSTMs were created expressly to address the issue of long-term reliance. They don't have to work hard to remember knowledge for lengthy periods of time; it's like second nature to them.

A sequence of repeating neural network modules make up all recurrent neural networks. In typical RNNs, his repeating module will have a single tanh layer as its structure, which is quite straightforward.

Although LSTMs have a chain-like structure, the repeating module differs. There are four neural network layers instead of one, and they all interact differently.

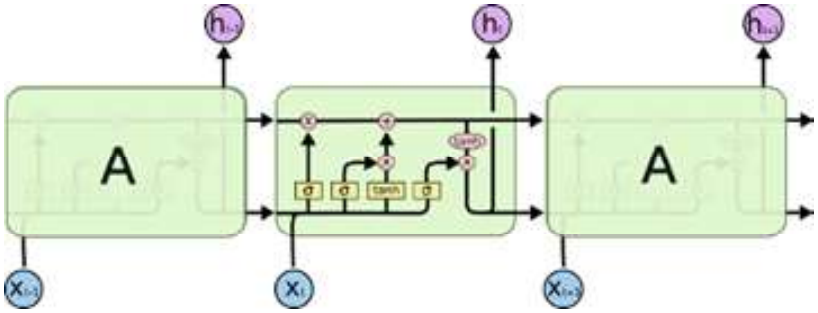

Figure 19: The Repeating Module in an LSTM Contains Four Interacting Layers

5.4.1. The Fundamental Concept Behind LSTMs

The key to LSTMs is the cell state, represented by the horizontal line at the top of the picture.

The state of the cell is comparable to that of a conveyor belt. It moves directly down the entire chain with only a few minor linear interactions. Data can very easily simply and unalteredly move over it.

The cell state, which is meticulously managed by structures known as gates, can be altered by the LSTM by deleting or adding information.

Information can be passed through gates in a controlled manner. They are composed of a sigmoid neural net layer and a pointwise multiplication step.

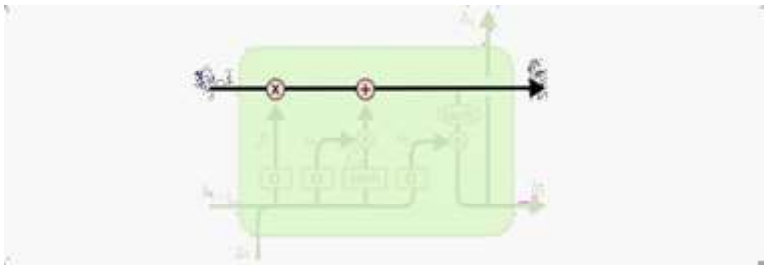

Figure 20: Cell State of the LSTM

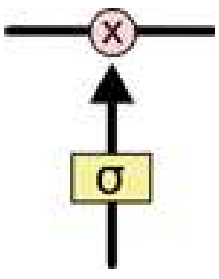

Figure 21: Gate of the LSTM

The sigmoid layer generates values from 0 to 1 that represent the amount of each component that should be permitted to pass. "Let everything through!" denotes a value of one, while "Allow nothing through!" denotes a value of zero.

6. Error Metrics

6.1. Root Mean Squared Error (RMSE)

It is a commonly used metric for comparing the predicted values (sample or population values) with the values observed by a model or estimator. The RMSE is defined as the square root of the second sample moment of the differences between predicted and observed values, or the quadratic mean of these differences. When the computations are performed over the data sample that was used for estimate, these deviations are referred to as residuals, and when they are computed out-of-sample, they are referred to as errors (or prediction errors). The RMSE is always positive, and a value of 0 (which is nearly never reached in practice) indicates that the data is perfectly suited. A lower RMSE is often preferable than a greater one. However, because the measure is dependent on the scale of the numbers used, comparisons between different types of data would be invalid.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \overline{x_i}\right)^2}{N}} \text{ where}$$

$RMSE$ = Root Mean Squared Error

N = The number of data points

x_i = Actual observations

$\overline{x_i}$ = Predicted observations

6.2. Mean Absolute Error (MAE)

It is a metric for comparing errors between paired observations describing the same occurrence.

$$MAE = \frac{\sum_{i=1}^{N}\left|x_i - \overline{x_i}\right|}{N} \text{ where}$$

MAE = Mean Absolute Error

N = The number of data points

x_i = Actual observations

$\overline{x_i}$ = Predicted observations

6.3. Mean Absolute Percentage Error (MAPE)

It is a statistical gauge of how accurately a forecasting technique predicts the future. When x_i and $\overline{x_i}$ represent the actual and projected values, respectively, the accuracy is often stated as a ratio determined by the formula. Divided by the actual value of x_i is the difference between them. The absolute value of this ratio is multiplied by the number of fitted points, n, for each projected point in time.

$$MAPE = \frac{\sum_{i=1}^{N}\left|\frac{x_i - \overline{x_i}}{x_i}\right|}{N} \text{ where}$$

$MAPE$ = Mean Absolute Percentage Error

N = The number of data points

x_i = Actual observations

$\overline{x_i}$ = Predicted observations

7. Results of ARIMA/Random Forest/Support Vector/ANN/CNN/LSTM Models

This section presents experimental results on the performance of the various models like ARIMA/Random Forest/Support Vector/ANN/CNN/LSTM for all the 8 datasets.

As can be seen from the above tables, the values of the three Error Metrics have been shown for all the 8 datasets. Picking the forecast with the smaller error measurement based on one of the Error Metrics is the obvious course of action. However, we must go a step further and decide whether this difference is important (for forecasting purposes) or merely the result of the sample's particular selection of data values.

Table 3: Error Metrics for ARIMA

	RMSE	Mean Absolute Error	MAPE
BSE Sensex	2166.97	1959.70	3.30
Govt Bond Yield	0.49	0.39	5.58
CPI	3.16	2.58	2.27
Exchange Rate	2.26	1.71	2.52
USGDP	2041.21	1842.22	10.10
US Corporate Bond Yield	0.41	0.33	11.16
USCPI Urban	5.46	4.27	1.60
US Interest Rate	1.66	1.34	106.45

Table 4: Error Metrics for Random Forest

	RMSE	Mean Absolute Error	MAPE
BSE Sensex	3746.93	3405.81	5.70
Govt Bond Yield	0.26	0.19	10.02
CPI	18.32	15.01	12.04
Exchange Rate	4.62	3.79	5.91
USGDP	4277.39	3438.73	17.60
US Corporate Bond Yield	0.62	0.45	20.77
USCPI Urban	18.07	15.39	5.77
US Interest Rate	0.34	0.24	147.92

Table 5: Error Metrics for Support Vector

	RMSE	Mean Absolute Error	MAPE
BSE Sensex	868.48	721.69	3.35
Govt Bond Yield	0.21	0.15	10.38
CPI	0.82	0.61	10.23
Exchange Rate	1.15	0.83	6.10
USGDP	420.58	173.06	16.48
US Corporate Bond Yield	0.16	0.12	21.99
USCPI Urban	0.53	0.39	4.08
US Interest Rate	0.27	0.13	157.37

Table 6: Error Metrics for ANN

	Train RMSE	Test RMSE	Train Mean Absolute Error	Test Mean Absolute Error	Train MAPE	Test MAPE
BSE Sensex	679.52	655.27	524.33	508.64	1.16	0.86
Govt Bond Yield	0.68	0.31	0.42	0.25	5.40	3.71
CPI	0.95	1.78	0.84	1.51	2.29	1.27
Exchange Rate	2.01	2.11	1.37	1.63	2.90	2.33
USGDP	48.36	386.69	32.96	173.87	1.29	0.96
US Corporate Bond Yield	0.64	0.24	0.55	0.19	8.58	6.12
USCPI Urban	0.55	0.76	0.43	0.60	0.22	0.23
US Interest Rate	0.57	0.42	0.47	0.27	21.81	52.71

Table 7: Error Metrics for LSTM

	Train RMSE	Test RMSE	Train Mean Absolute Error	Test Mean Absolute Error	Train MAPE	Test MAPE
BSE Sensex	636.22	621.02	488.86	479.85	1.08	0.81
Govt Bond Yield	1.02	0.29	0.72	0.23	7.75	3.43
CPI	0.68	1.04	0.59	0.85	1.53	0.72
Exchange Rate	1.19	1.27	0.77	0.96	1.63	1.38
USGDP	78.84	431.79	55.76	201.22	2.03	1.11
US Corporate Bond Yield	0.30	0.25	0.24	0.21	4.39	6.54
USCPI Urban	0.56	0.90	0.46	0.64	0.23	0.24
US Interest Rate	0.33	0.33	0.22	0.21	10.30	38.89

Table 8: Error Metrics for CNN

	Train RMSE	Test RMSE	Train Mean Absolute Error	Test Mean Absolute Error	Train MAPE	Test MAPE
BSE Sensex	614.15	641.54	462.66	475.66	1.03	0.81
Govt Bond Yield	0.55	0.30	0.34	0.23	4.55	3.48
CPI	0.81	1.57	0.63	1.28	1.51	1.11
Exchange Rate	1.30	1.40	0.90	1.06	1.91	1.52
USGDP	46.65	422.62	31.81	192.56	1.29	1.07
US Corporate Bond Yield	0.47	0.23	0.39	0.18	6.20	5.79
USCPI Urban	1.57	1.89	1.47	1.75	0.75	0.67
US Interest Rate	0.41	0.41	0.31	0.21	15.90	45.49

To evaluate whether the two forecasts are significantly different, we run the Diebold-Mariano test.

8. Results of the Diebold Mariano Test

BSE

Table 9: Diebold Mariano Result for BSE

Model 1	Model 2	p value	Result
ARIMA	ANN	0.00008	Significant Difference
ARIMA	CNN	0.000004	Significant Difference
ARIMA	LSTM	0.00007	Significant Difference
ARIMA	Random Forest	1.999	No Significant Difference
ARIMA	Support Vector	1.999	No Significant Difference
Random Forest	Support Vector	1.0005	No Significant Difference
Random Forest	ANN	1.999	No Significant Difference
Random Forest	CNN	1.999	No Significant Difference
Random Forest	LSTM	1.999	No Significant Difference
Support Vector	ANN	0.999	No Significant Difference
Support Vector	CNN	0.158	No Significant Difference
Support Vector	LSTM	0.0906	No Significant Difference
ANN	CNN	0.9614	No Significant Difference
ANN	LSTM	0.084	No Significant Difference
CNN	LSTM	0.902	No Significant Difference

Exchange Rate

Table 10: Diebold Mariano Result for Exchange Rate USGDP

Model 1	Model 2	p value	Result
ARIMA	ANN	0.97	No Significant Difference
ARIMA	CNN	0.338	No Significant Difference
ARIMA	LSTM	0.29	No Significant Difference
ARIMA	Random Forest	1.92	No Significant Difference
ARIMA	Support Vector	1.90	No Significant Difference
Random Forest	Support Vector	1.99	No Significant Difference
Random Forest	ANN	1.96	No Significant Difference
Random Forest	CNN	1.98	No Significant Difference
Random Forest	LSTM	1.98	No Significant Difference
Support Vector	ANN	1.99	No Significant Difference
Support Vector	CNN	1.97	No Significant Difference
Support Vector	LSTM	1.95	No Significant Difference
ANN	CNN	0.001	Significant Difference
ANN	LSTM	0.0009	Significant Difference
CNN	LSTM	0.185	No Significant Difference

USGDP

Table 11: Diebold Mariano Result for USGDP

Model 1	Model 2	p value	Result
ARIMA	ANN	0.0000004	Significant Difference
ARIMA	CNN	0.0000003	Significant Difference
ARIMA	LSTM	0.0000004	Significant Difference
ARIMA	Random Forest	1.97	No Significant Difference
ARIMA	Support Vector	1.99	No Significant Difference
Random Forest	Support Vector	1.98	No Significant Difference
Random Forest	ANN	1.97	No Significant Difference
Random Forest	CNN	1.98	No Significant Difference
Random Forest	LSTM	1.98	No Significant Difference
Support Vector	ANN	1.99	No Significant Difference
Support Vector	CNN	1.99	No Significant Difference
Support Vector	LSTM	1.97	No Significant Difference
ANN	CNN	0.66	No Significant Difference
ANN	LSTM	1.93	No Significant Difference
CNN	LSTM	1.82	No Significant Difference

Govt. Bond Yield

Table 12: Diebold Mariano Result for Govt. Bond Yield

Model 1	Model 2	p value	Result
ARIMA	ANN	0.23	No Significant Difference
ARIMA	CNN	0.25	No Significant Difference
ARIMA	LSTM	0.24	No Significant Difference
ARIMA	Random Forest	0.06	No Significant Difference
ARIMA	Support Vector	1.97	No Significant Difference
Random Forest	Support Vector	1.99	No Significant Difference
Random Forest	ANN	0.0007	Significant Difference
Random Forest	CNN	0.001	Significant Difference
Random Forest	LSTM	0.006	Significant Difference
Support Vector	ANN	1.99	No Significant Difference
Support Vector	CNN	1.99	No Significant Difference
Support Vector	LSTM	1.99	No Significant Difference
ANN	CNN	0.90	No Significant Difference
ANN	LSTM	1.04	No Significant Difference
CNN	LSTM	1.15	No Significant Difference

US Corporate Bond Yield

Table 13: Diebold Mariano Result for US Corporate Bond Yield

Model 1	Model 2	p value	Result
ARIMA	ANN	0.11	No Significant Difference
ARIMA	CNN	0.09	No Significant Difference
ARIMA	LSTM	0.21	No Significant Difference
ARIMA	Random Forest	1.85	No Significant Difference
ARIMA	Support Vector	1.99	No Significant Difference
Random Forest	Support Vector	1.96	No Significant Difference
Random Forest	ANN	1.92	No Significant Difference
Random Forest	CNN	1.92	No Significant Difference
Random Forest	LSTM	1.89	No Significant Difference
Support Vector	ANN	1.99	No Significant Difference
Support Vector	CNN	1.99	No Significant Difference
Support Vector	LSTM	1.99	No Significant Difference
ANN	CNN	0.40	No Significant Difference
ANN	LSTM	1.37	No Significant Difference
CNN	LSTM	1.55	No Significant Difference

US Interest Rate

Table 14: Diebold Mariano Result for US Interest Rate

Model 1	Model 2	p value	Result
ARIMA	ANN	0.015	Significant Difference
ARIMA	CNN	0.015	Significant Difference
ARIMA	LSTM	0.011	Significant Difference
ARIMA	Random Forest	0.009	Significant Difference
ARIMA	Support Vector	1.99	No Significant Difference
Random Forest	Support Vector	1.95	No Significant Difference
Random Forest	ANN	0.275	No Significant Difference
Random Forest	CNN	0.378	No Significant Difference
Random Forest	LSTM	0.336	No Significant Difference
Support Vector	ANN	1.89	No Significant Difference
Support Vector	CNN	1.81	No Significant Difference
Support Vector	LSTM	1.95	No Significant Difference
ANN	CNN	0.535	No Significant Difference
ANN	LSTM	0.225	No Significant Difference
CNN	LSTM	0.430	No Significant Difference

USCPI Urban

Table 15: Diebold Mariano Result for USCPI Urban

Model 1	Model 2	p value	Result
ARIMA	ANN	0.043	Significant Difference
ARIMA	CNN	0.058	No Significant Difference
ARIMA	LSTM	0.038	Significant Difference
ARIMA	Random Forest	1.99	No Significant Difference
ARIMA	Support Vector	1.96	No Significant Difference
Random Forest	Support Vector	1.99	No Significant Difference
Random Forest	ANN	1.99	No Significant Difference
Random Forest	CNN	1.99	No Significant Difference
Random Forest	LSTM	1.99	No Significant Difference
Support Vector	ANN	1.98	No Significant Difference
Support Vector	CNN	2.00	No Significant Difference
Support Vector	LSTM	1.88	No Significant Difference
ANN	CNN	1.99	No Significant Difference
ANN	LSTM	1.31	No Significant Difference
CNN	LSTM	0.37	No Significant Difference

CPI

Table 16: Diebold Mariano Result for CPI

Model 1	Model 2	p value	Result
ARIMA	ANN	0.77	No Significant Difference
ARIMA	CNN	0.019	Significant Difference
ARIMA	LSTM	0.03	Significant Difference
ARIMA	Random Forest	1.99	No Significant Difference
ARIMA	Support Vector	1.99	No Significant Difference
Random Forest	Support Vector	1.99	No Significant Difference
Random Forest	ANN	1.99	No Significant Difference
Random Forest	CNN	1.99	No Significant Difference
Random Forest	LSTM	1.99	No Significant Difference
Support Vector	ANN	1.99	No Significant Difference
Support Vector	CNN	1.99	No Significant Difference
Support Vector	LSTM	1.99	No Significant Difference
ANN	CNN	0.01	Significant Difference
ANN	LSTM	0.000005	Significant Difference
CNN	LSTM	1.35	No Significant Difference

9. Graphical Analysis

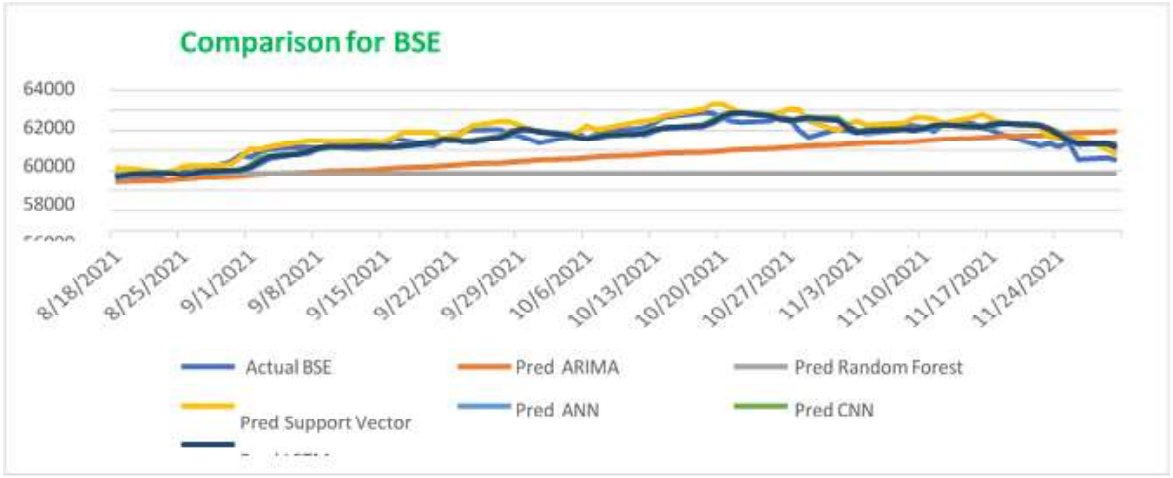

Figure 22: Comparison of Different Methods for BSE

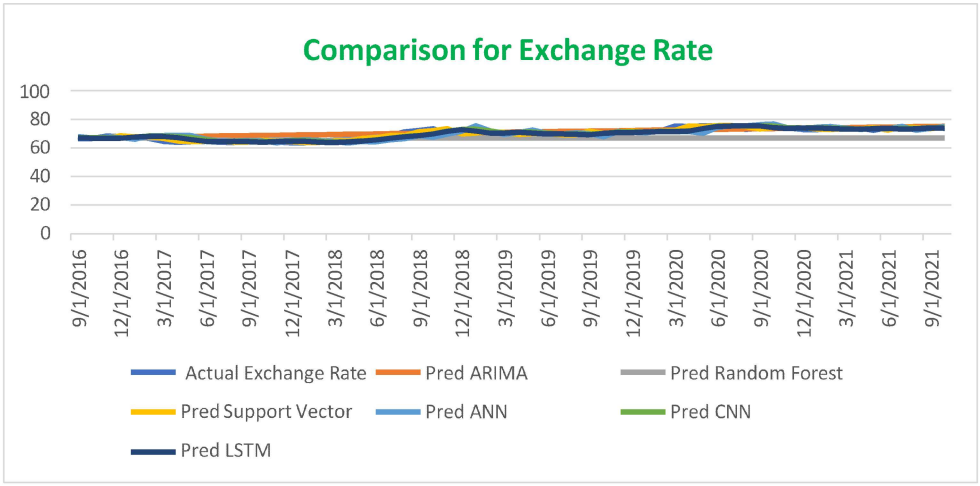

Figure 23: Comparison of Different Methods for Exchange Rate

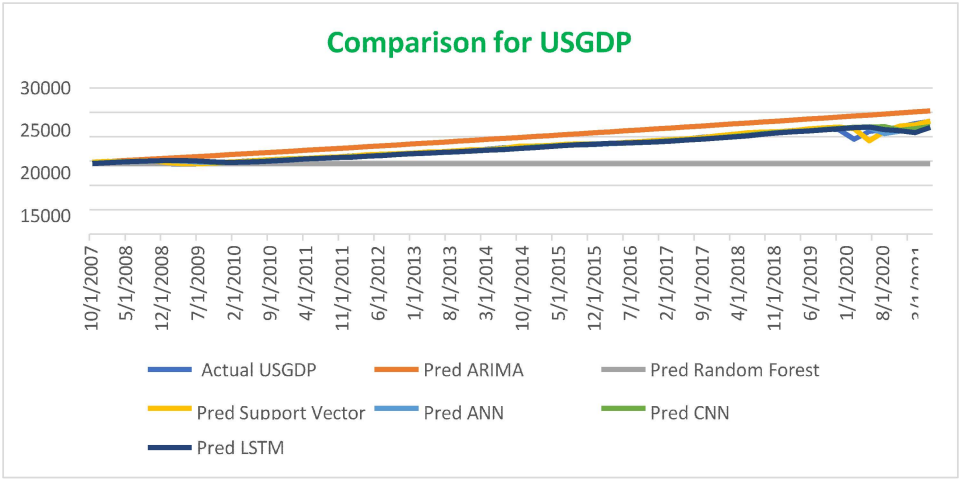

Figure 24: Comparison of Different Methods for USGDP

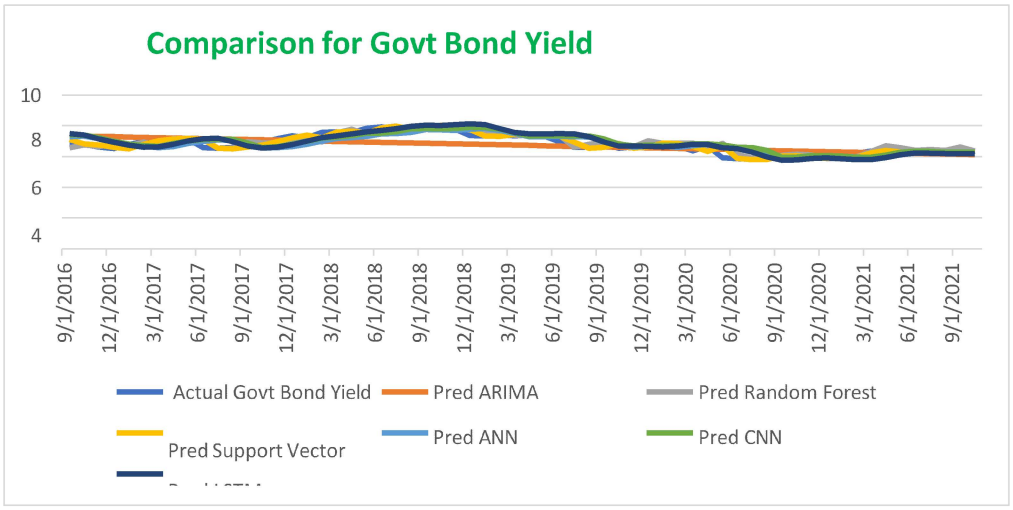

Figure 25: Comparison of Different Methods for Govt. Bond Yield

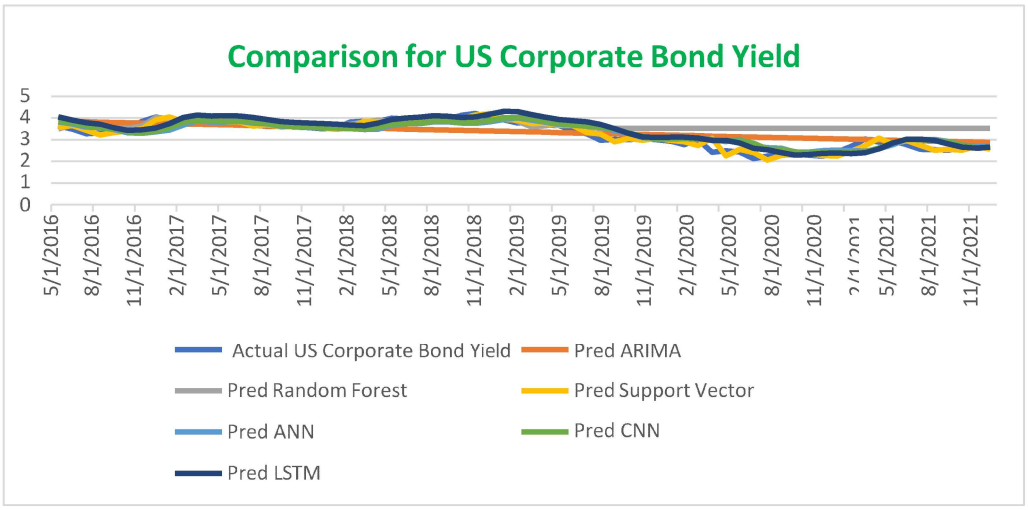

Figure 26: Comparison of Different Methods for US Corporate Bond Yield

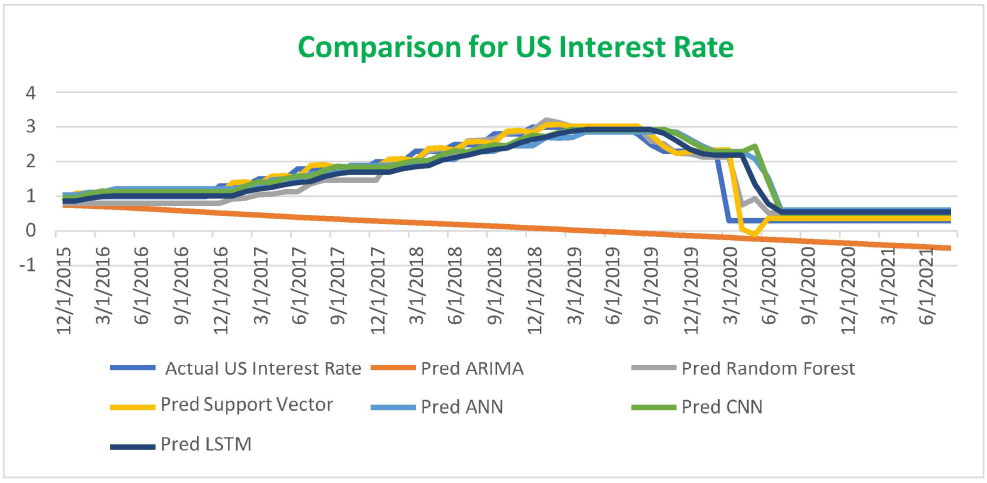

Figure 27: Comparison of Different Methods for US Interest Rate

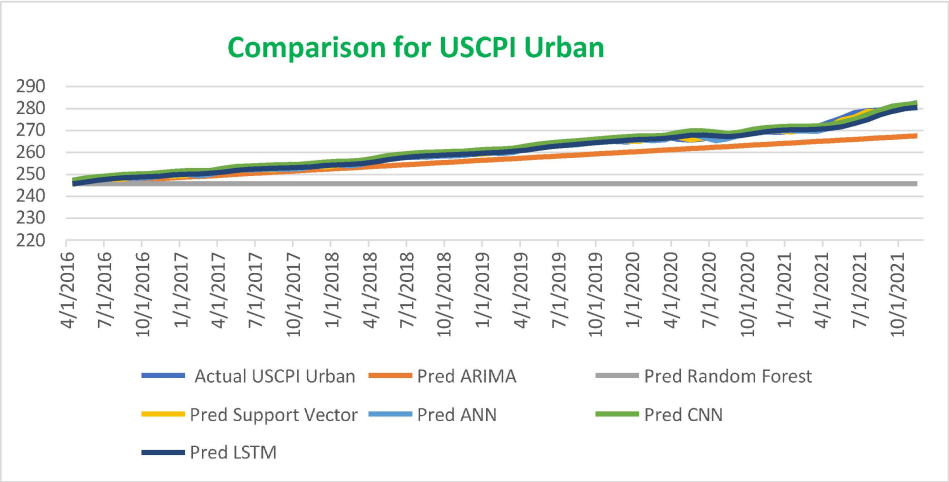

Figure 28: Comparison of Different Methods for USCPI Urban

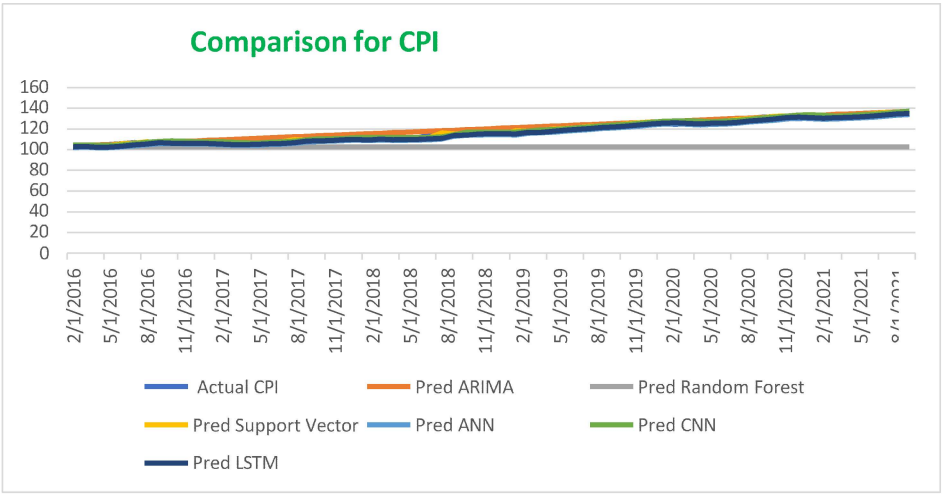

Figure 29: Comparison of Different Methods for CPI

10. Summary and Conclusion

This paper compares the performance of the Statistical Model ARIMA, Machine Learning Models i.e Random Forest and Support Vector Machine and Deep Learning Models, i.e., ANN, CNN and RNN(LSTM) for the 8 datasets namely BSE Indices (India), Govt Bond Yield (India), CPI (India), Exchange Rate (INR - USD), GDP of US (USA), US Corporate Bond Yield, USCPI Urban (USA) and US Interest Rate (USA). The data has been collected from the Economic Research, Federal Reserve Bank of St. Louis. All the data are macroeconomic indicators as well as of univariate time series.

For the forecasting models, again different patterns have emerged. I have restricted myself to one step forecasting. I have used the Diebold Mariano test to compare the forecasts from the different models. For most of the datasets, the Deep Learning Models outperform the standard ARIMA model whereas for one of the datasets, there is no significant difference between the forecasts. Accuracy of the forecasts can be enhanced by combination of various methods like XGBoost or AdaBoost and one can dive into these models in future.

References

Ballings, Michel, Dirk Van den Poel, Nathalie Hespeels and Ruben Gryp. (2015). Evaluating Multiple Classifiers for Stock Price Direction Prediction. *Expert Systems with Applications*, 42, 7046-7056.

Basak Suryoday, Saibal Kar, Snehanshu Saha, Luckyson Khaidem and Sudeepa Roy Dey (2019). Predicting the Direction of Stock Market Prices Using Tree-Based Classifiers. *NA Journal of Economics and Finance,* 47, 552-567.

Chen, W.H., Shih, J.Y. and Wu, S. (2006). Comparison of Support-Vector Machines and Back Propagation Neural Networks in Forecasting the Six Major Asian Stock Markets. *International Journal of Electronic Finance,* 1, 49-67.

Christoffersen, Peter F. and Francis X. Diebold (2006). Financial Asset Returns, Direction-of Change Forecasting, and Volatility Dynamics. NBER Working Paper No. w10009.

Elman, J.L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2), 179-211.

Gray Wesley and Jack Vogel (2016). Quantitative Momentum: A Practitioner's Guide to Building a Momentum-Based Stock Selection System. John Wiley & Sons, Hoboken.

Hochreiter and Schmidhuber (1997). Long Short Term Memory. *Neural Computation*, 9(8), 1735-1780.

Huang, W., Nakamori, Y. and Wang, S.Y. (2005). Forecasting Stock Market Movement Direction with Support Vector Machine. *Computers and Operations Research*, 32(10), 2513-2522.

Jordan, M.I. (1990). Artificial Neural Network. IEEE Press.

Kim, K.J. (2003). Financial Time Series Forecasting Using Support Vector Machines. *Neurocomputing*, 55(1-2), 307-319.

Lo Andrew W., Harry Mamaysky and Jiang Wang (2000). Foundations of Technical Analysis: Computational Algorithms, Statistical Inference, and Empirical Implementation. *The Journal of Finance*, 55, 1705-1765.

Lohrmann Christoph and Pasi Luukka (2019). Classification of Intraday S&P500 Returns with a Random Forest. *International Journal of Forecasting,* 35, 390-407.

Malkiel Burton G. (2003). The Efficient Market Hypothesis and its Critics. *Journal of Economic Perspectives,* 17, 59-82.

Mokoaleli-Mokoteli, Thabang, Shaun Ramsumar and Hima Vadapalli (2019). The Efficiency of Ensemble Classifiers in Predicting the Johannesburg Stock Exchange All-Share Index Direction. *Journal of Financial Management, Markets and Institutions*, 7, 1950001.

Moskowitz Tobias J., Yao Hua Ooi and Lasse Heje Pedersen (2012). Time Series Momentum. *Journal of Financial Economics,* 104, 228-250.

Nti Isaac Kofi, Adebayo Felix Adekoya and Benjamin Asubam Weyori (2020). A Comprehensive Evaluation of Ensemble Learning for Stock-Market Prediction. *Journal of Big Data,* 7, 20.

Nyberg Henri and Harri Pönkä (2016). International Sign Predictability of Stock Returns: The Role of the United States. *Economic Modelling,* 58, 323-338.

Nyberg, Henri (2011). Forecasting the Direction of the US Stock Market with Dynamic Binary Probit Models. *International Journal of Forecasting,* 27, 561-578.

Pai, P.-F. and Lin, C.-S. (2005). A Hybrid ARIMA and Support Vector Machines Model in Stock Price Forecasting. *Omega*, 33, 497-505.

Tay, F.E.H. and Cao, L.J. (2001). Improved Financial Time Series Forecasting by Combining Support Vector Machines with Selforganizing Feature Map. *Intelligent Data Analysis*, 5, 339-354.

Tay, F.E.H. and Cao, L.J. (2002). Modifed Support Vector Machines in Financial Time Series Forecasting. *Neurocomputing*, 48, 847-861.

Weng Bin, Lin Lu, Xing Wang, Fadel M. Megahed and Waldyn Martinez (2018). Predicting Short-Term Stock Prices Using Ensemble Methods and Online Data Sources. *Expert Systems with Applications*, 112, 258-73.

Cite this article as: Krishnandu Ghosh (2024). A Comparison of Standard Statistical, Machine Learning and Deep Learning Methods in Forecasting the Time Series. *International Journal of Artificial Intelligence and Machine Learning,* 4(2), 106-133. doi: 10.51483/IJAIML.4.2.2024.106-133.