CHAPTER IV

OPTIMIZING EXTREME LEARNING MACHINE ALGORITHM WITH DETERMINISTIC WEIGHT MODIFICATION FOR PREDICTING STOCK PRICE USING HISTORICAL DATA

4.1 Introduction

Stock exchanges, market data providers, financial data suppliers, and research platforms are frequently the sources of these historical statistics. They can be applied to several different tasks, such as machine learning-based prediction models, algorithmic trading, and quantitative, technical, and fundamental analysis. Ensuring the quality, accuracy, and completeness of data is crucial when utilizing historical information for analytical and research purposes. In terms of historical data, we choose five features. Including the opening, the highest, lowest, closing price, `and the adjusted close.

- **Opening price:** The price at which a stock is employed for the initial time at the start of an exchange session is branded as the opening price of that stock. The dynamics of supply and demand during the pre-market and opening auction stages impact this pricing. Because it establishes the stock's starting valuation for the trading day. The opening price is significant.

- **Highest price:** the highest price can be accomplished on the stock market is the maximum price it can trade at for a given amount of time. Frequently, one trading day stands for the highest price that purchasers were prepared to pay for the stock at that specific point in the period.

- **Lowest price:** the lowest price that a detailed stock can trade at during a given time frame that is mentioned to as the lowest price in the stock market. It stands for the lowest price that consumers were arranged to pay for the stock during that period.

- **Closing price:** the last price at which a specific stock is traded on a given trading day is referred to as the closing price. It is the final price that is recorded before the market closes for the trading day. For traders and investors. The closing price is a crucial piece of information since it expresses the mood of the market and the stock's ultimate valuation at the end of the trading day.

- **Adjusted close:** The cash value of the final traded price before the market shuts. The adjusted closing price includes anything that may affect the stock price after the market closes for the day. A stock price is often influenced by the supply and demand of market participants.

4.2 ELM

The first phase is to use the optimized ELM approach-based DWM which is used historical datasets for stock prediction. The details of ELM and DWM methods are discussed in the following sections. An input, an output, and a hidden layer with a sizable number of nodes for nonlinear processing comprise the basic three-layer structure of ELM, an SLFN. Figure 3 displays the ELM framework. Network properties are fixed, the hidden nodes are launched at random, and connection weights need to be learned. The benefits of ELM over FNN have been demonstrated on a variety of tasks across several fields [66]. Its foundation is experimental risk reduction, where the weight at the output layer is characterized by initial random weight production and biases [67]. On the other hand, it offers faster learning at a higher level of effectiveness.

At a quicker learning rate, it offers more efficient generalization, though. The data models $x_i \in R^n$ and the goal value $t_i \in R^m$for N training samples (x_i, t_i). The following is the mathematical expression for ELM:

$$o_i = \sum_{i=1}^{L} w_i \, g_i(w_i . x_j + b_i), j = 1,2,..,N \tag{5}$$

Where, w_i - weights between hidden and input layer. β_i - weight for layers of output and hidden. $b_i \in \mathbb{R}$ - bias. $g(x)$ activation functions. o_i – output layer. L - number of hidden neurons. The N examples can be assessed with zero error. i. e $\sum_{i=1}^{N} \|z_i - y_i\| = 0$. If $\beta_i, w_i$and $b_j$exist such that,

$$Hw = o \tag{6}$$

Where,

$$H = \begin{pmatrix} g(w_1 . x_1 + b_1) & \cdots & g(w_H . x_1 + b_H) \\ \vdots & .. & \vdots \\ g(w_1 . x_N + b_1) & \dots & g(w_H . x_N + b_H) \end{pmatrix}_{N \times H} \tag{7}$$

The output layer is calculated as below,

$O = [o_1, o_{2,\ldots,} o_N]^T$ and $w = [w_1, w_{2,\ldots,} w_L]^T$. H - hidden layer output matrix. The existence of w_i indicates the model with hidden nodes (L) can learn training samples (N), so that,

$$t_j = \sum_{i=1}^{L} w_i \, g_i(w_i . x_j + b_i), j = 1,2,..,N \quad (8)$$

Where, t_j is the target vector. The following is a compact representation of the aforementioned equation in matric vector form:

$$Hw = t \quad (9)$$

$t_j = [t_1, t_2, \ldots, t_N]$ is the target vector. Because the input weights and hidden layer bias were chosen at random at the beginning of learning, the preceding equation becomes a linear parameter system. The linear constraint system's least norm least squares solution is

$$w = H \dagger t \quad (10)$$

where $H \dagger$ is the Moore–Penrose generalized inverse of H. To solve the least squares issue effectively and get the output layer weights, the inverse matrix operation is necessary. Compared to standard neural networks that use iterative optimization procedures, it enables ELM to achieve quick training speed. Furthermore, because the inverse matrix produces a unique solution without changing weights dependent on training data, it reduces the likelihood of overfitting in ELM. Some of the remarkable points are discussed as follows,

- **SLFNs:** SLFN designs are the foundation of ELMs. ELMs randomly initialize the hidden layer weights and fix them during training, in contrast to standard neural networks, where the input and hidden layer weights are learned repeatedly through training. In a single data pass, just the output weights are learned.

- **Hidden layer weights:** The hidden layer neurons' weights are initialized at random. However, ELMs can accomplish faster training times and calculating efficiency because they do not require iterative optimization measures like gradient descent because of their random initialization.

- **Fixed hidden layer weights:** The initialization weights of the neurons in the hidden layer remain constant during training. The fixed-hidden layer weights feature of ELMs sets them apart from other ANN architectures and facilitates training while consuming less processing resources.

- **Single-step learning process:** In ELMs, a single-step learning procedure is used to train the output weights, typically with linear regression or other regression techniques. Fast training times and efficient learning are achieved by understandably learning the output weights without the requirement for backpropagation or iterative optimization approaches.
- **Universal approximators:** ELMs are called universal function approximators because, given enough hidden neurons, they can approximate any continuous function with any precision. This feature makes ELMs useful for a range of regression and classification problems, such as time series prediction, pattern recognition, and control systems.
- **Applications:** Several fields, such as pattern recognition, image processing, signal processing, and financial forecasting, have seen the successful application of ELMs. Because of its computational efficiency and capacity to handle high-dimensional data, ELMs have been utilized in finance for risk management, portfolio optimization, and stock market prediction.

Algorithm 1: ELM

Step 1: Randomly initialize the input weights and hidden biases

Step 2: Calculate the hidden output matrix H

Step 3: Estimate the $H^{\dagger}$ as the MP generalized inverse obtained from H

Step 4: Calculate the output weights matrix $\hat{\beta}$

- **Advantages:** ELMs' primary benefits are their simplicity, computational economy, and quick training timeframes. Significant data processing jobs that need efficiency and speed, like actual applications, big data analytics, and online knowledge scenarios, are particularly compatible with ELMs.
- **Limitations:** ELMs have assistance, but they also have certain drawbacks. For example, when the random initialization causes a subpar routine, the fixed-hidden layer weights may yield less-than-ideal results. Additionally, ELMs might not be as actual as more advanced ANN architectures when it comes to tasks that call for representation learning.

4.3 Deterministic weight modification (DWM)

In ML, DWM is a specific technique wherein changes to the weights of ANNs are

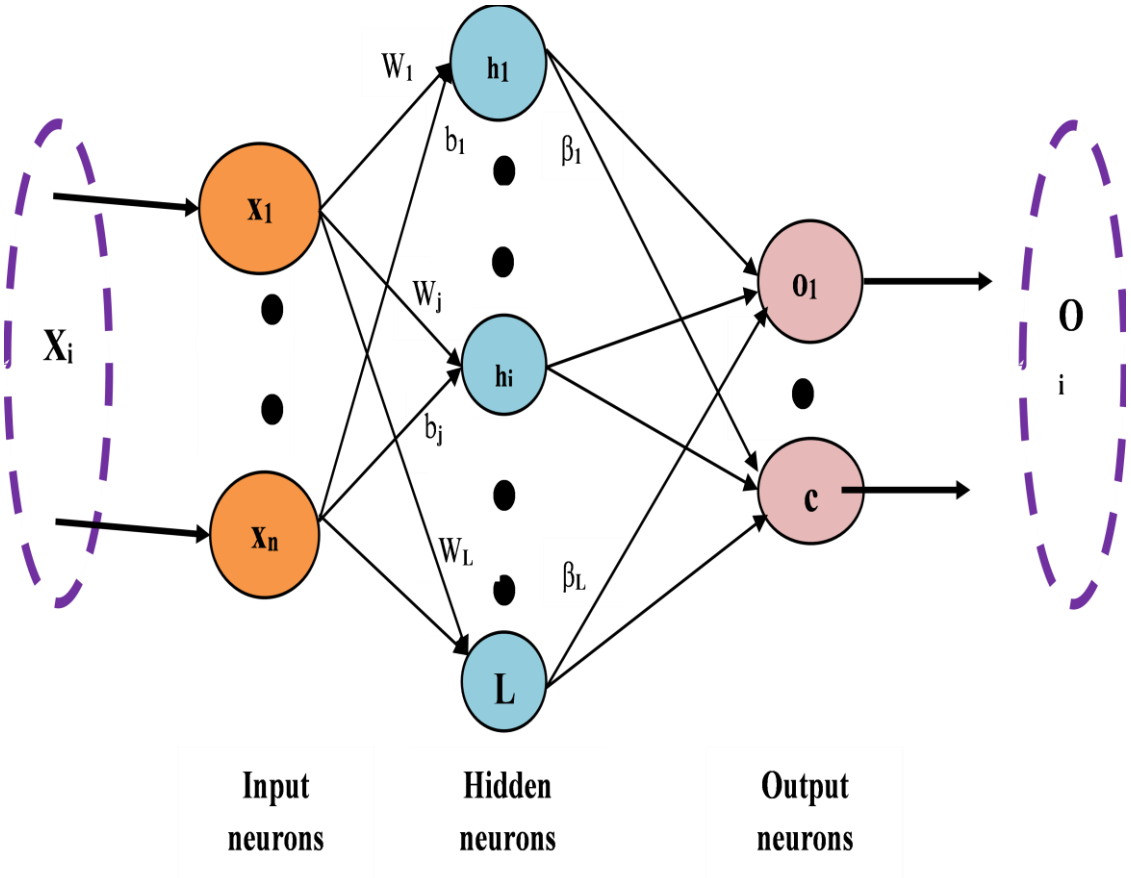

Figure 3 : Framework of ELM

detailed instead of being learned iteratively from the data during training. Weight adjustment in classic ANN training usually includes iterative optimization approaches, like gradient descent, where the weights are changed in tiny steps according to the discrepancy between the goal principles and the model's forecasts.

This process continues until the performance of the training data spreads to a suitable level. DWM predefines the adjustments to the weights based on some specified criteria or rule, as different from altering the weights iteratively based on the training data. This method can minimize training and processing complexity when the optimization problem has a closed-form solution or when the relationship between the inputs and outputs is explicit.

The DWM is used to adjust the weights and biases of the ELM to expand prediction accuracy and generalization efficiency. DWM seeks to modify the weights and biases during the learning process. It can locate the global solution with the least error signal and the fastest convergence time. The goal is to minimize system error by deterministically adjusting the weights and bias to get outputs that are almost identical to the desired outcome [21, 68-70].

In machine learning and optimization, deterministic weight changes are weight update procedures that have no stochastic or random components and instead follow a fixed, predetermined pattern. These changes guarantee that the model will always converge to the same set of weights while the initial circumstances and input data remain the same. Exact mathematical rules are used to modify weights without adding noise. In machine learning models, deterministic weight adjustments offer a reliable and methodical learning strategy. Although they work well for convex and small-scale problems, they can be computationally expensive for large datasets. Optimizing model performance in a variety of machine learning applications requires knowing when to employ deterministic versus stochastic approaches.

The network is given a weight modification to obtain the outputs this close to the desired output. The learning process has retreated to identify the matching modification in the weights because the predictable outcome for the future iteration is predetermined. Since the reverse computation is excessively inefficient, the solution is found using specific approximations. Compared to the current iteration, the output after this one will probably be somewhat similar to what was expected following the modification.

Algorithm 2: DWM

Step 1: **While (converged)**

Step 2: $\Delta E = [E(i) - E(i-t]/t$

Step 3: If $(\Delta E \leq G_T)$ and $(E(i) - E_T)$ then

Step 4: For all p

$$E_p(i+1) = \frac{1}{2}\sum_{m=1}^{M}(t_{pm} - o_{pm}(i+1))]^2$$

$$p^* = \text{minimum}(E_p(i+1))$$

Step 5: End for

Step 6: m^* is selected randomly y between M and 1

Step 7: For all k

$$k^* = \text{minimum}(\min(\bar{o}_{p^*k}(i), 1 - \bar{o}_{p^*k}(i))$$

Step 8: End for

Step 9: If $t_{p^*m^*=1}$ then

$$o_{p^*m^*}(i+1) = 1 - \sqrt{2\lambda E(i)/PM}$$

Step 10: Else

$$o_{p^*m^*}(i+1) = \sqrt{2\lambda E(i)/PM}$$

Step 11: End if

Step 12: $\Delta\bar{o}_{p*k*}(i+1) = \frac{(f^{-1}(o_{p*m*}(i+1)) - \sum_{k=1}^{K} w_{km*}(i+1)o_{p*k*}(i))}{w_{k*m*}(i+1)}$

Step 13: $\bar{o}_{p*k*}(i+1) = \Delta\bar{o}_{p*k*}(i+1) + \bar{o}_{p*k*}(i)$

Step 14: $\varepsilon_{p*k*} = f^{-1}(\bar{o}_{p*k*}(i+1) - f^{-1}(\bar{o}_{p*k*}(i)$

Step 15: For all n

$$\Delta\bar{w}_{nk*}(i+1) = \frac{\bar{\varepsilon}_{p*k*}}{\sum_{n=1}^{N} x_{p*n}^2}$$

Step 16: End for

Step 17: End if

Step 18: **End while**

$$E_O(i) = \frac{1}{2}\sum_{p=1}^{P}[t_{po} - y_{po}(i)]^2 \geq \frac{E(i)}{O} \quad (11)$$

The goal of adjusting the weights is to get this output closer to the target output, which is described as follows:

$$y_{PO}(i+1) = y_{PO}(i) + \beta\big(t_P(i) - y_{PO}(i)\big) = \beta t_{PO}(i) + (1-\beta)O_{PO}z \quad (12)$$

Where, $0 < \beta < 1$ for all data samples $'P'$. The output is the weights for the subsequent iteration derived from the preceding equations,

$$y_{pm}(i+1) \;=\; f\left(\sum_{k=1}^{K} w_{ko}(i+1)\;\bar{y}_{pk}(i+1)\right) \quad (13)$$

And thus,

$$\sum_{k=1}^{K} \Delta w_{ko}(i+1)\;\bar{y}_{pk}(i+1) = \varepsilon_{po} \quad (14)$$

Where,

$$\varepsilon_{po} = f^{-1}\left(y_{po}(i+1)\right) - \sum_{k=1}^{K} w_{ko}(i)\;\bar{y}_{pk}(i+1) \quad (15)$$

$$f^{-1}(x) = ln\left(\frac{x}{1-x}\right) \quad (16)$$

So, weight is calculated $\Delta w_{ko}(i+1)$ as follows,

$$\Delta w_{ko}(i+1) = \frac{\sum_{P=1}^{P} e_{po}\,\bar{y}_{Pk}(i+1)}{\sum_{P=1}^{P} \bar{y}_{pk}^2(i+1)} \quad (17)$$

For each hidden neuron, both $\Delta w_{ko}(i+1)$ and error are calculated as follows,

$$E_o^{(k)}(i+1) = \frac{1}{2}\sum_{p=1}^{P}\sum_{m=1}^{M}\big[t_{po} - y_{po}(i+1)\big]^2 \quad (18)$$

An error is defined as a selection that is larger than the average error. The error is then completely reduced by manipulating the output layer results. The target in the output neurons is t_{pm}, and the expected output is $y_{PO}(i) + \beta\big(t_P(i) - y_{PO}(i)\big)$. β is too close to one, which could lead to pointless actions. The goal value will be the predictable output when β is set to 1. It is hard to demonstrate that the system can achieve the intended outcome in a single cycle.

Algorithm 3: DWM-ELM

Step 1: Randomly initialize the input weights and hidden biases

Step 2: Calculate the hidden-layer output H

Step 3: Estimate the $H^{\dagger}$ as the MP generalized inverse obtained from H

Step 4: Compute the output weights matrix $\hat{\beta}$ using DWM (Algorithm 2)

Compared to iterative optimization algorithms, deterministic weight modification techniques require less hyperparameter tuning. As a result, there is less chance of parameter selection bias and the training procedure is simplified. Comparing deterministic weight modification techniques to iterative optimization techniques, the former can assist minimize overfitting by offering a unique solution to the output layer weights. The architecture of DWM-ELM is shown in Figure 4.

4.4 Compared methods

The present research work and its strengths are compared with some variants of ELM, SVM, and BPNN and the following subsection discusses SVM and BPNN. When the dataset is limited or interpretability is crucial, SVM is an excellent option for stock price prediction. Selecting the appropriate kernel allows it to be especially helpful in capturing non-linear interactions.

Larger datasets and intricate patterns, on the other hand, are a strength of BPNN. Although it needs to be tuned carefully to prevent overfitting, it is more adaptable and can learn a broad variety of interactions between various characteristics. A detailed explanation of both SVM and BPNN is given below.

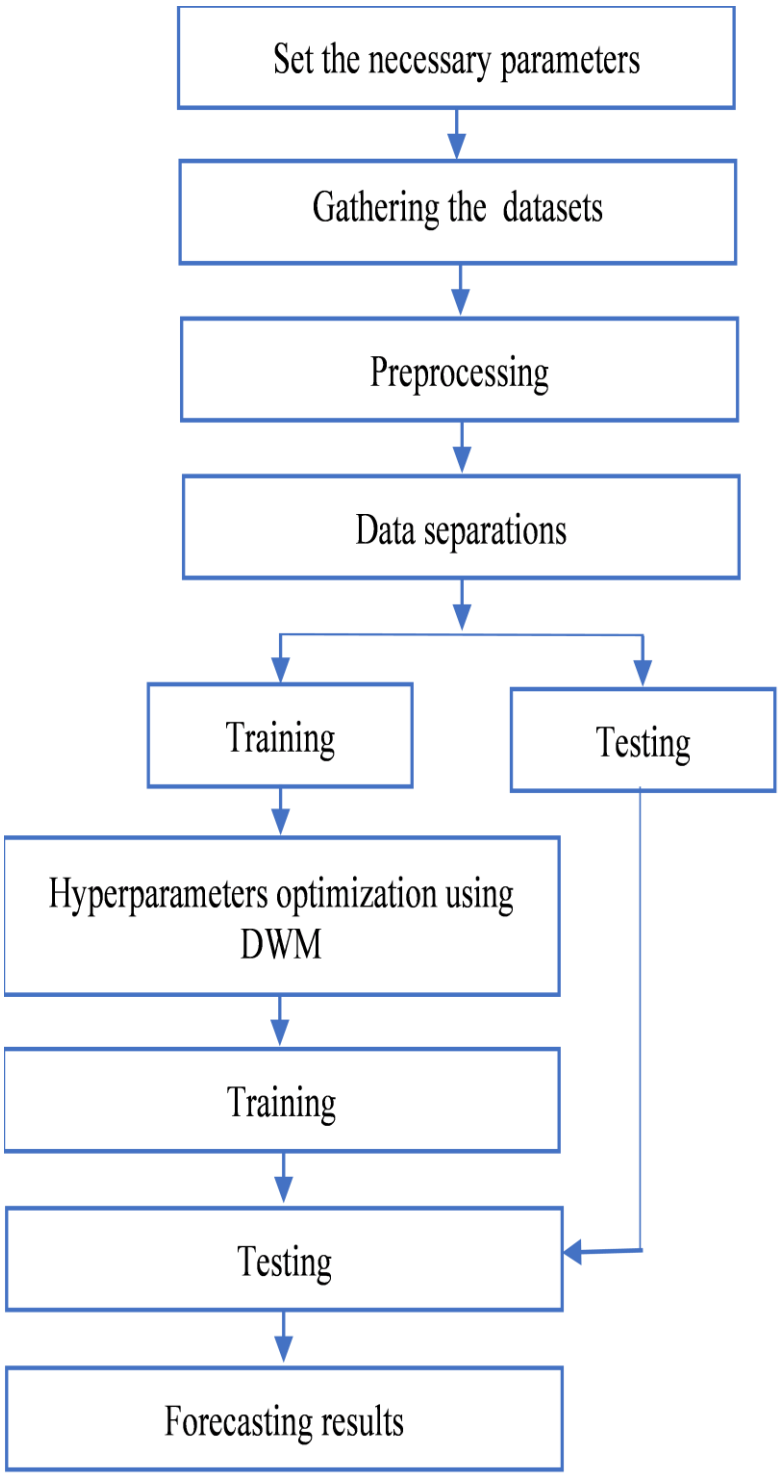

Figure 4 : DWM-ELM for stock price prediction

4.4.1 SVM

SVM is a cutting-edge ML method for regression analysis and classification in the statistical learning theory of multi-dimensional functions. It can approximate any multivariate function universally and to any desired degree of precision. With high accuracy, SVM has been applied in most engineering domains [71]. Three key components are most important in SVM Hyperplane, support vectors, and margin**.**

- **Hyperplane:** A hyperplane is a line that divides data into two classes in two dimensions. It is a flat affine subspace in higher dimensions. Finding the hyperplane that maximizes the margin—that is, the distance between the hyperplane and the closest data point for any class—is the aim of SVM.

- **Support vectors:** The data samples that are nearest to the hyperplane are these. They are essential for defining the hyperplane's location. Support vectors form the foundation of the SVM model, and eliminating any more data points leaves the hyperplane in the same location.

- **Margin:** The distance between the nearest data point from either class and the hyperplane is known as the margin. The classifier's generalization improves with increasing margin.

4.4.2 BPNN algorithm

BPNN is a type of ANN that uses a supervised learning method to train the network. It is a foundational algorithm in the field of DL and is widely used in various applications. BPNN contains an input, more than one hidden, and an output layer. Numerous neurons make up each layer and the weights involved in the connections between the neurons are altered through training.

- **Feedforward:** The feedforward process includes the propagation of input layer by layer across the network. A sum of inputs with weights is followed by the output of each neuron. The output of each layer is used as the input for the layer after it up until the final output is produced.

- **Error computations:** Once the output is attained, the error between the forecast output and the true aim output is measured using a loss or cost function.

- **Backpropagation:** Backpropagation refers to the process of updating the weights of the connections to minimize the error between the predicted and the true target output. This process involves propagating the error backward through the network, from the output to the input layer, and adjusting the weights using gradient descent optimization.

- **Gradient Descent:** Gradient descent is the optimization approach used to update the connection weights. The connection weights must be adjusted in the opposite direction of the gradient to minimize the loss which is resolute by calculating the gradient of the loss function concerning each weight. BPNN has recognized unexpected success in various domains. However, they also have boundaries, such as sensitivity to hyperparameters. Despite these challenges, BPNN remains a controlling and extensively used tool.

4.5 Performance measures

The accuracy and efficacy of stock market prediction models in predicting future stock prices or trends are evaluated through performance criteria. Performance indicators offer important information about how robust, accurate, and reliable stock market prediction algorithms are. To fully assess model performance, many indicators may be utilized in combination or prioritized based on the particular goals and specifications of the prediction task. The performance is measured with five performance metrics namely, MSE, RMSE, MAE, MAPE, and R- Square (R^2) .

4.6 Experimental Result and Analysis

The results analysis of Phase 1 of this study is for predicting the stock market price which uses historical features such as one particular stock day's open, low, high, closing, and adjusted closing prices as inputs to the proposed deterministic weight modification based Extreme learning machine algorithm referred as DELM-H. The DWM method is used to connect weights and bias of ELM for enhancing the performance. The present sub-section discusses the ability of the developed DELM-H method. Tables 2, 4, 6, 8, and 10 show the training performance of MSE, RMSE, MAE, MAPE, and R-Square values, respectively.

Tables 3, 5, 7, 9, and 11 show the testing performance of MSE, RMSE, MAE, MAPE, and R-Square values respectively. Figures 5, 7, 9, 11, and 13, show the training performance of MSE, RMSE, MAE, MAPE, and R-Square values, respectively. Figures 6, 8, 10, 12, and 14, show the testing performance of MSE, RMSE, MAE, MAPE, and R-Square values respectively.

In the training phase, the suggested technique obtained low MSE values when compared to other prediction algorithms. Such as 0.000738, 0.000129, 0.000875, 0.000758, 0.000691, and 0.000653 for S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively. In comparison to previous related prediction algorithms. The recommended technique produced low RMSE values: 0.03240, 0.07628, 0.02607, 0.02190, 0.02429, and 0.02469 for the S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively.

In comparison to other related prediction algorithms. The recommended technique produced low MAE values: 0.0079,0.0084, 0.0051, 0.0058, 0.0063, and 0.0067 for S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively. In comparison to other evaluated prediction algorithms. The recommended technique produced low MAPE values: 0.0024, 0.0079, 0.0046, 0.0008, 0.0057 and 0.0059 for S&P, Nifty 50, SBIN, ICICI, HDFC and MSFT respectively. When compared to other prediction algorithms, the recommended technique obtained high R-Square values: 0.9437, 0.9437, 0.9572, 0.9408, 0.9672, 0.9461, 0.9494 and S&P, Nifty 50, SBIN, ICICI, HDFC and MSFT, respectively. In the testing phase, the suggested technique obtained low MSE values when compared to other prediction algorithms, the recommended technique produced low MSE values: 0.00105, 0.00582, 0.00068, 0.00048, 0.00059, and 0.00061 for S&P, Nifty 50, SBIN, ICICI, HDFC and MSFT, respectively.

In comparison to previous related prediction algorithms. The recommended technique produced low RMSE values: 0.02716, 0.01135, 0.02958, 0.02753, 0.02628, and 0.02555 for the S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively. In comparison to other related prediction algorithms. The recommended technique produced low MAE values: 0.0102, 0.0752, 0.0119, 0.0122, 0.0138, and 0.0143 for S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively. In comparison to other evaluated prediction algorithms. The recommended technique produced low MAPE values of 0.0068, 0.0053, 0.0031, 0.0037, 0.0039, and 0.0061 for S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively. When compared to other prediction algorithms. The recommended technique obtained high R-Square values: 0.9437, 0.9601, 0.9785, 0.9495, 0.9782, and 0.9467 and S&P, Nifty 50, SBIN, ICICI, HDFC, and MSFT, respectively.

Table 2 : Training Performance Comparisons for MSE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.000738	0.000843	0.001291	0.00482	0.006293	0.009061
Nifty 50	0.000129	0.001365	0.001517	0.001738	0.001995	0.002155
SBIN	0.000875	0.000937	0.001045	0.001591	0.001836	0.002067
ICICI	0.000758	0.000863	0.000983	0.001135	0.001455	0.001864
HDFC	0.000691	0.000821	0.000973	0.001847	0.002851	0.003195
MSFT	0.000653	0.000694	0.000792	0.000984	0.001587	0.001918

Table 3 : Testing Performance Comparisons for MSE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.00105	0.00122	0.00294	0.00346	0.00598	0.00718
Nifty 50	0.00582	0.00768	0.00892	0.01029	0.01529	0.01989
SBIN	0.00068	0.000865	0.000942	0.001191	0.001395	0.001759
ICICI	0.00048	0.000658	0.000893	0.001221	0.001426	0.001852
HDFC	0.00059	0.000798	0.000931	0.001194	0.001352	0.001698
MSFT	0.00061	0.00695	0.00749	0.00897	0.01099	0.0201

Table 4 : Training Performance Comparisons for RMSE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.02716	0.02903	0.03593	0.06942	0.07932	0.09518
Nifty 50	0.01135	0.03694	0.03894	0.04168	0.04466	0.04642
SBIN	0.02958	0.03061	0.03232	0.03988	0.04284	0.04546
ICICI	0.02753	0.02937	0.03135	0.03368	0.03814	0.04317
HDFC	0.02628	0.02865	0.03119	0.04297	0.05339	0.05652
MSFT	0.02555	0.02634	0.02814	0.03136	0.03983	0.04379

Table 5 : Testing performance comparisons for RMSE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.03240	0.03492	0.05422	0.05882	0.07733	0.08473
Nifty 50	0.07628	0.08763	0.09444	0.10144	0.12365	0.14103
SBIN	0.02607	0.02941	0.03069	0.03451	0.03735	0.04194
ICICI	0.02190	0.02565	0.02988	0.03494	0.03776	0.04303
HDFC	0.02429	0.02824	0.03051	0.03455	0.03677	0.04120
MSFT	0.02469	0.08336	0.08654	0.09471	0.10483	0.14177

Table 6: Training Performance Comparisons for MAE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.0079	0.0095	0.0106	0.0381	0.0451	0.0621
Nifty 50	0.0084	0.0092	0.0192	0.0281	0.041	0.0515
SBIN	0.0051	0.0059	0.00631	0.0072	0.0095	0.0103
ICICI	0.0058	0.0069	0.0081	0.0172	0.0377	0.0412
HDFC	0.0063	0.0072	0.0079	0.0085	0.0093	0.01016
MSFT	0.0067	0.0075	0.0084	0.0095	0.0101	0.0115

Table 7 : : Testing Performance Comparisons for MAE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.0102	0.0106	0.0119	0.0125	0.0141	0.0152
Nifty 50	0.0752	0.0886	0.0938	0.1022	0.1256	0.1495
SBIN	0.0119	0.0124	0.0142	0.0172	0.0199	0.0208
ICICI	0.0122	0.0153	0.0291	0.0315	0.0515	0.0821
HDFC	0.0138	0.0145	0.0156	0.0176	0.0181	0.0201
MSFT	0.0143	0.0127	0.0372	0.0511	0.0676	0.0859

Table 8: Training performance comparisons for MAPE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.0024	0.0035	0.0059	0.0069	0.0088	0.0105
Nifty 50	0.0079	0.0092	0.0102	0.0221	0.0281	0.0315
SBIN	0.0046	0.0056	0.0064	0.0071	0.009	0.011
ICICI	0.0008	0.0009	0.0018	0.0026	0.0039	0.0090
HDFC	0.0057	0.0068	0.0072	0.0079	0.0094	0.0101
MSFT	0.0059	0.0063	0.0071	0.0078	0.0099	0.0105

Table 9: Testing performance comparisons for MAPE

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.0068	0.0072	0.0080	0.0086	0.0098	0.0095
Nifty 50	0.0053	0.0065	0.0078	0.0086	0.0094	0.0105
SBIN	0.0031	0.0035	0.0051	0.0072	0.0099	0.0109
ICICI	0.0037	0.0041	0.0067	0.0098	0.0139	0.0193
HDFC	0.0039	0.0055	0.0069	0.0081	0.0095	0.0111
MSFT	0.0061	0.0065	0.0076	0.0081	0.0095	0.0112

Table 10 : Training performance comparisons for R-Square

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.9672	0.9467	0.9174	0.8999	0.8544	0.8197
Nifty 50	0.9461	0.9165	0.8954	0.8542	0.8192	0.7615
SBIN	0.9494	0.9154	0.8926	0.8762	0.8462	0.8164
ICICI	0.9572	0.9286	0.9051	0.8725	0.8246	0.7916
HDFC	0.9408	0.9358	0.9027	0.8726	0.8291	0.8165
MSFT	0.9437	0.9248	0.9027	0.871	0.8389	0.8034

Table 11 : Testing performance comparisons for R-Square

Methods	DELM-H	GAELM	KELM	ELM	SVM	BPNN
S & P	0.9437	0.9294	0.9027	0.8834	0.8564	0.8164
Nifty 50	0.9601	0.9468	0.9121	0.8598	0.8262	0.8062
SBIN	0.9785	0.9564	0.9256	0.8718	0.8302	0.7861
ICICI	0.9495	0.9387	0.8836	0.8591	0.8261	0.7864
HDFC	0.9782	0.9527	0.9293	0.8901	0.8519	0.7909
MSFT	0.9467	0.9325	0.9027	0.8852	0.8561	0.8259

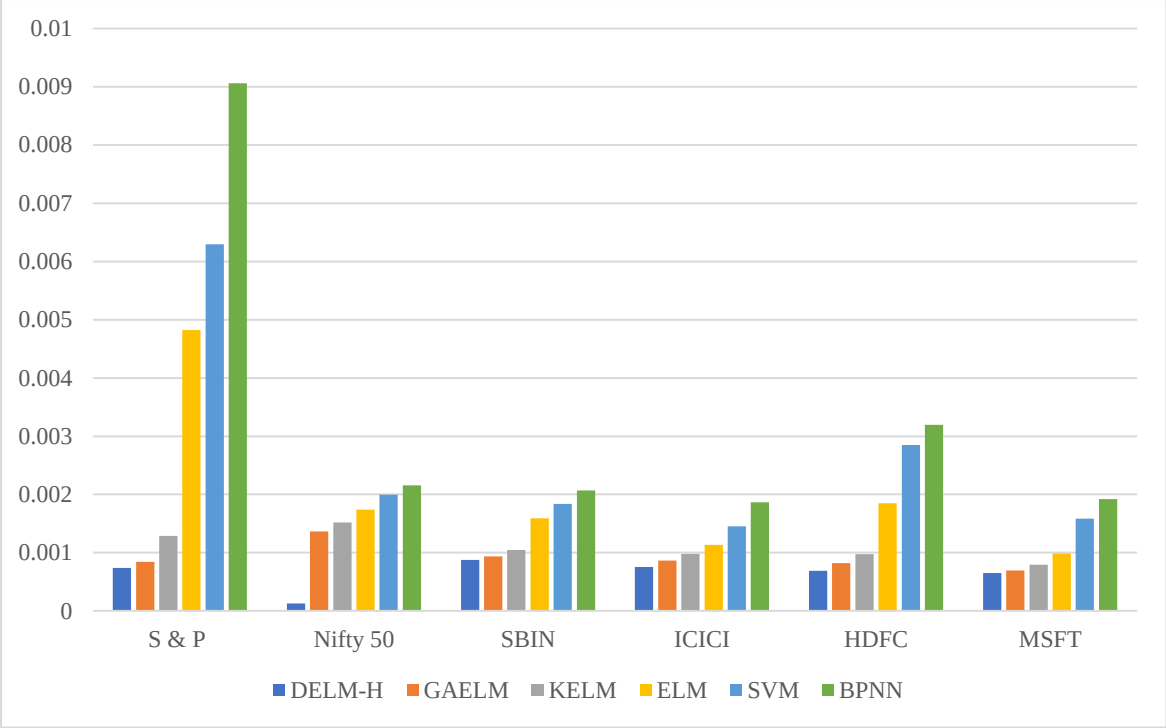

Figure 5 : Training Performance Comparisons for MSE

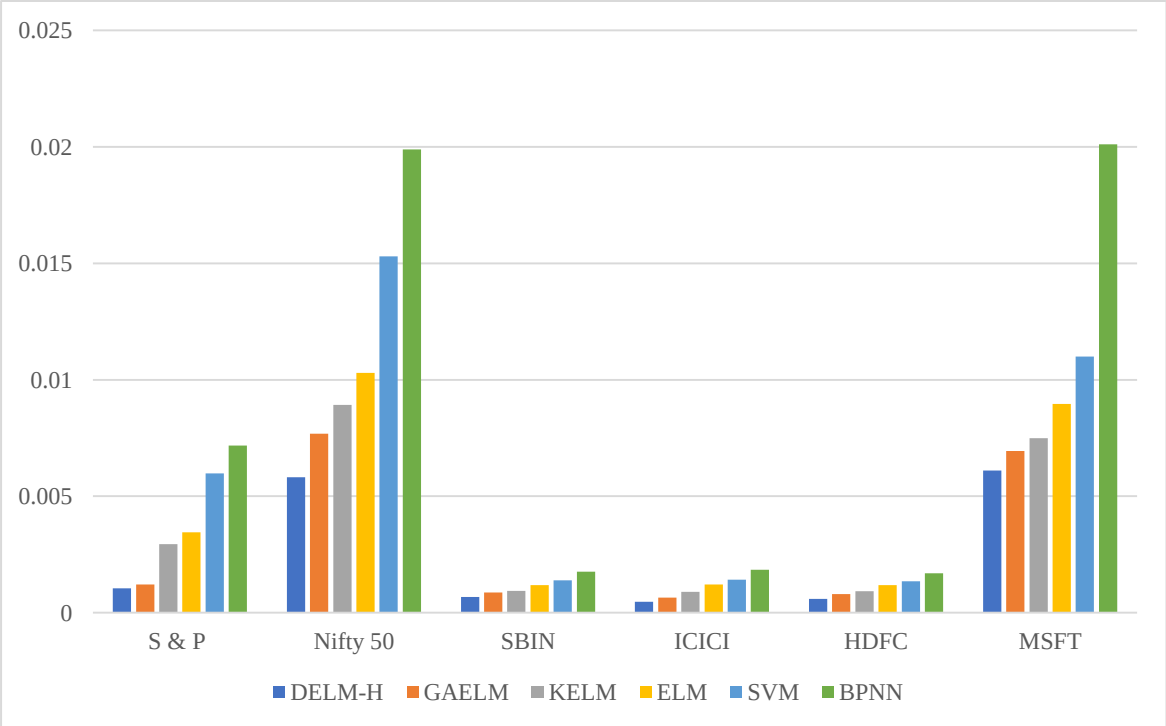

Figure 6 : Testing performance comparisons for MSE

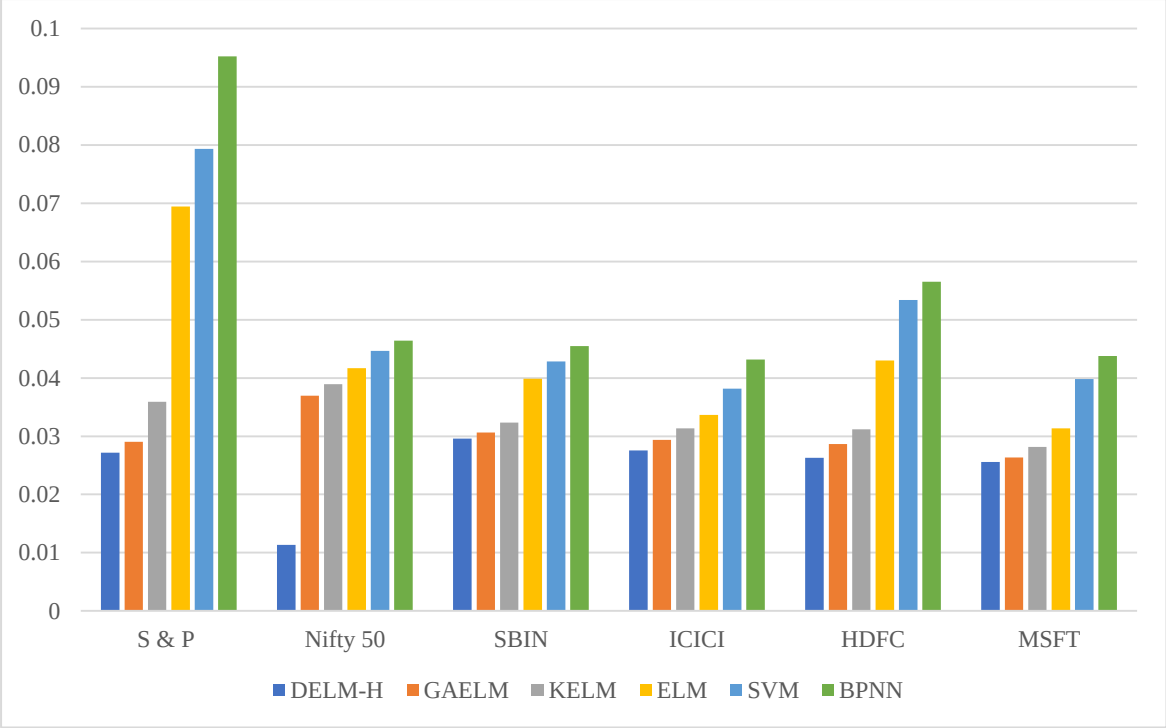

Figure 7 : Training performance comparisons for RMSE

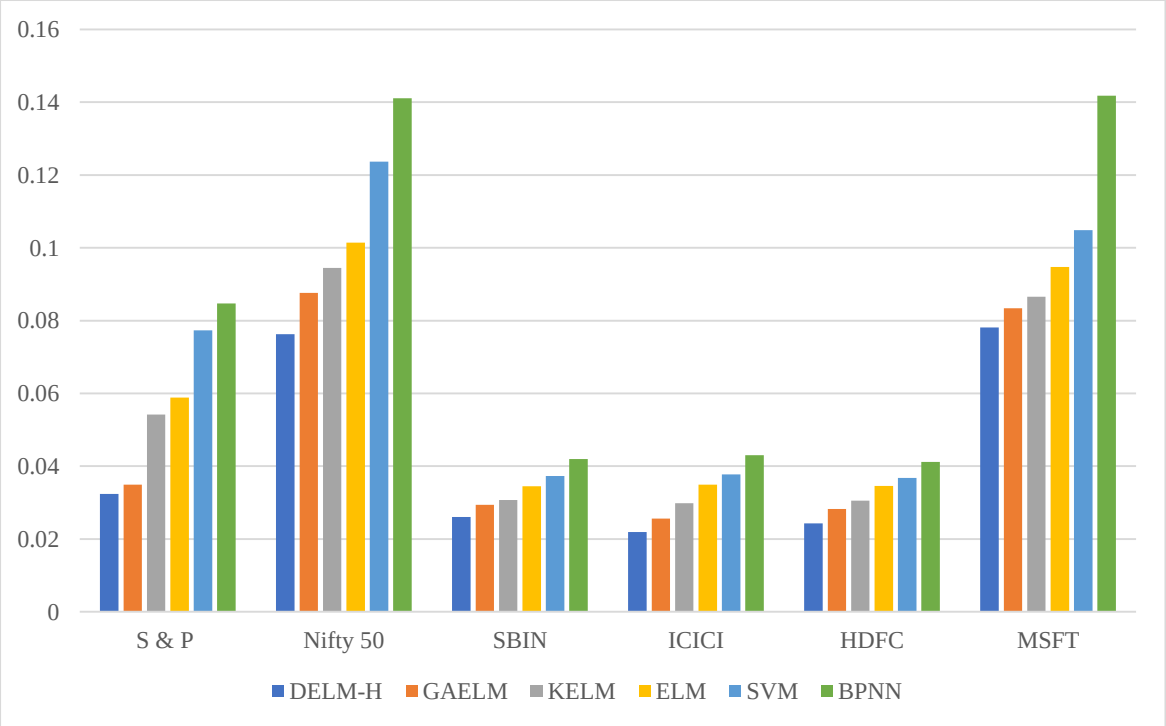

Figure 8: Testing Performance Comparisons for RMSE

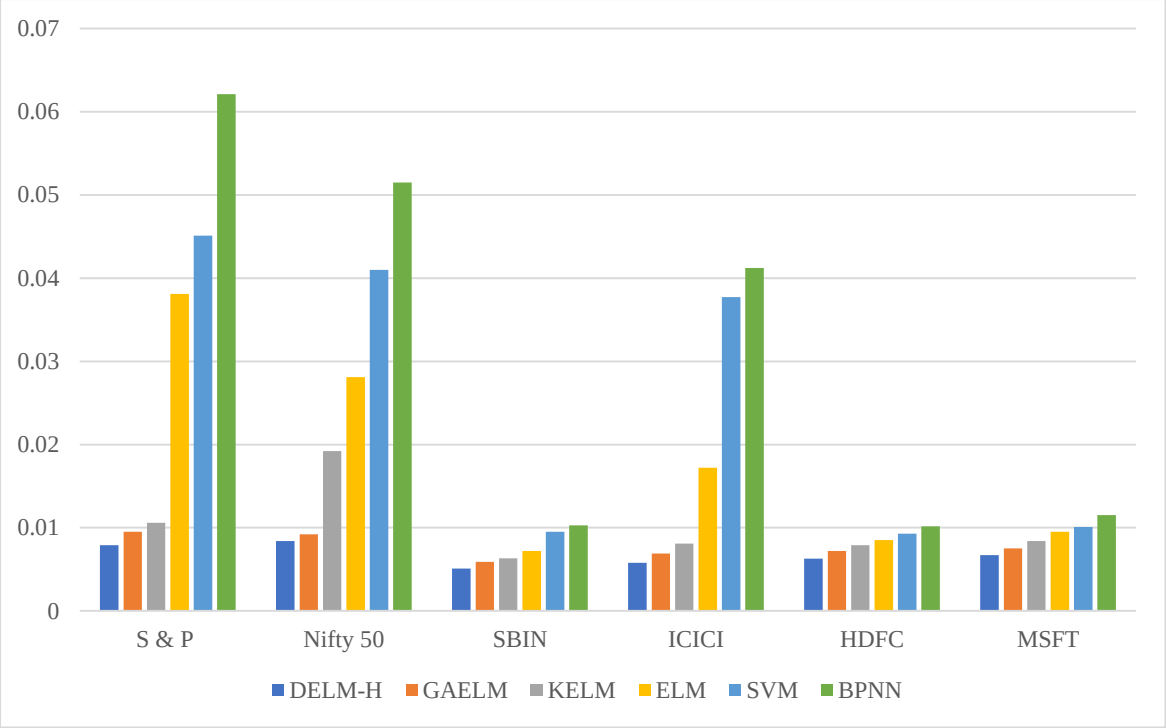

Figure 9 : Training performance comparisons for MAE

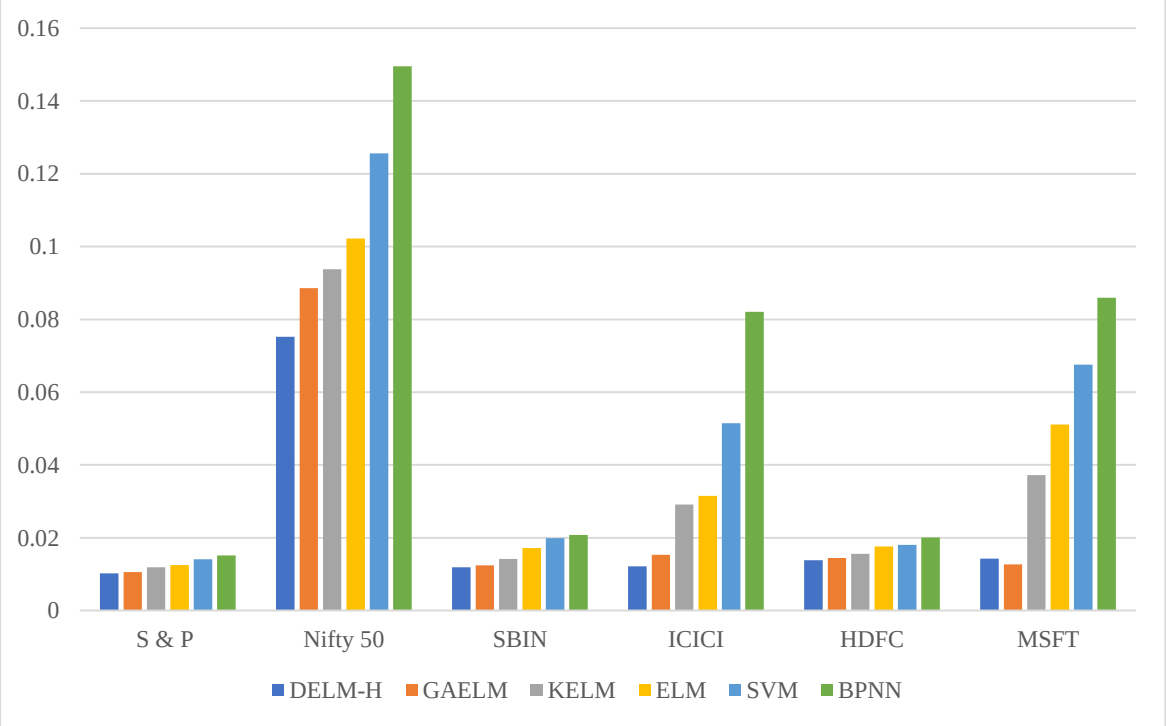

Figure 10 : Testing performance comparisons for MAE

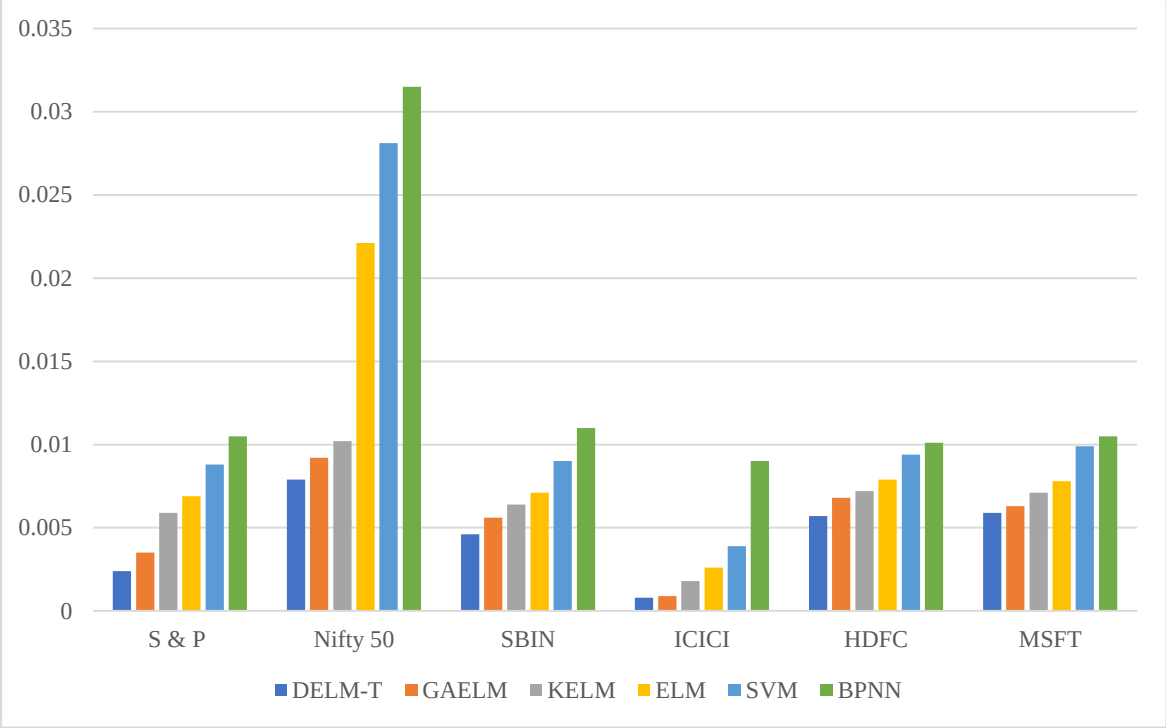

Figure 11: Training performance comparisons for MAPE

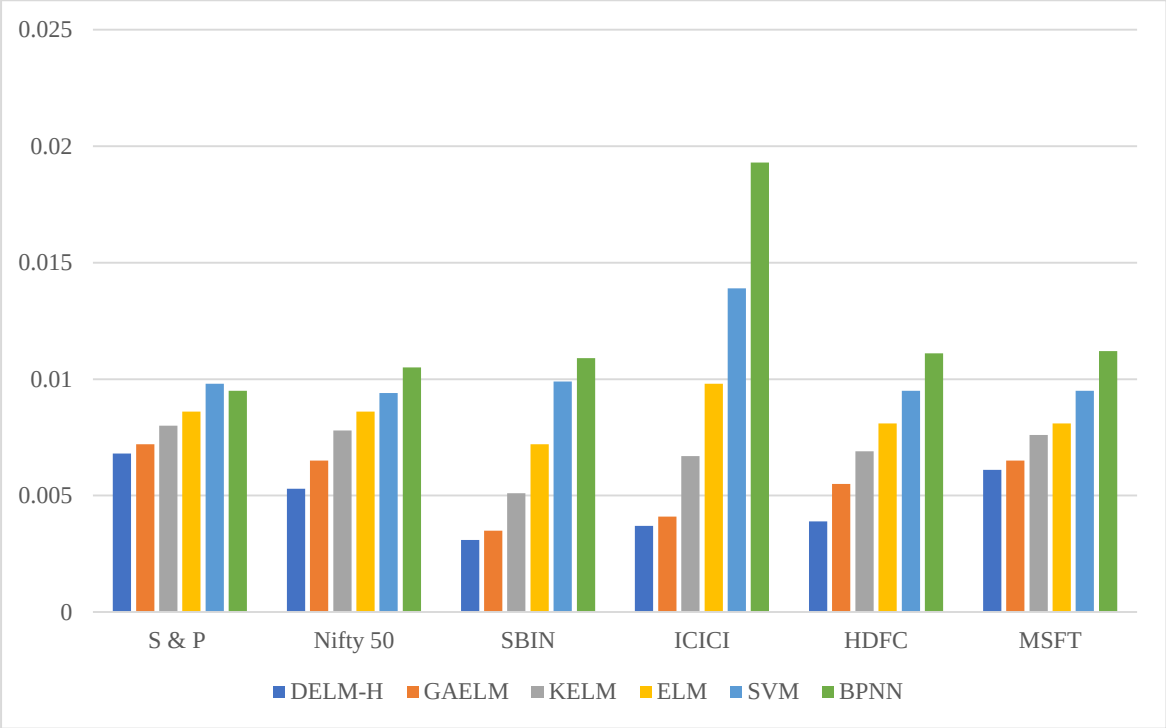

Figure 12 : Testing performance comparisons for MAPE

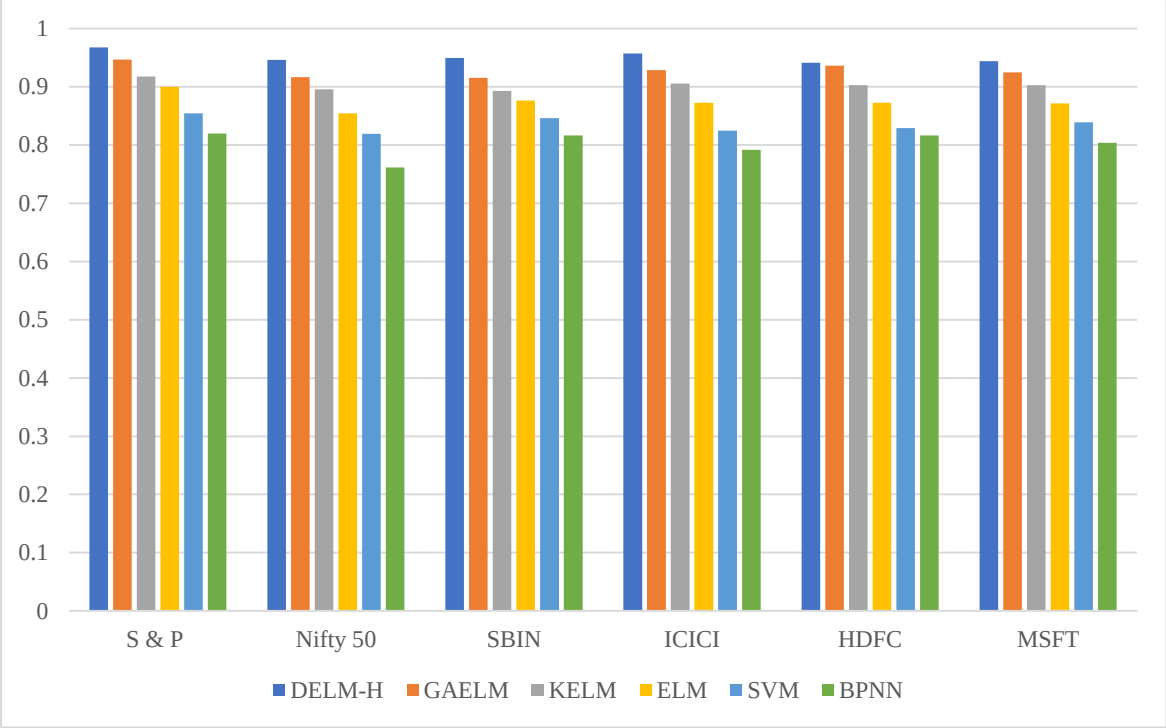

Figure 13*:* Training performance comparisons for R-Square

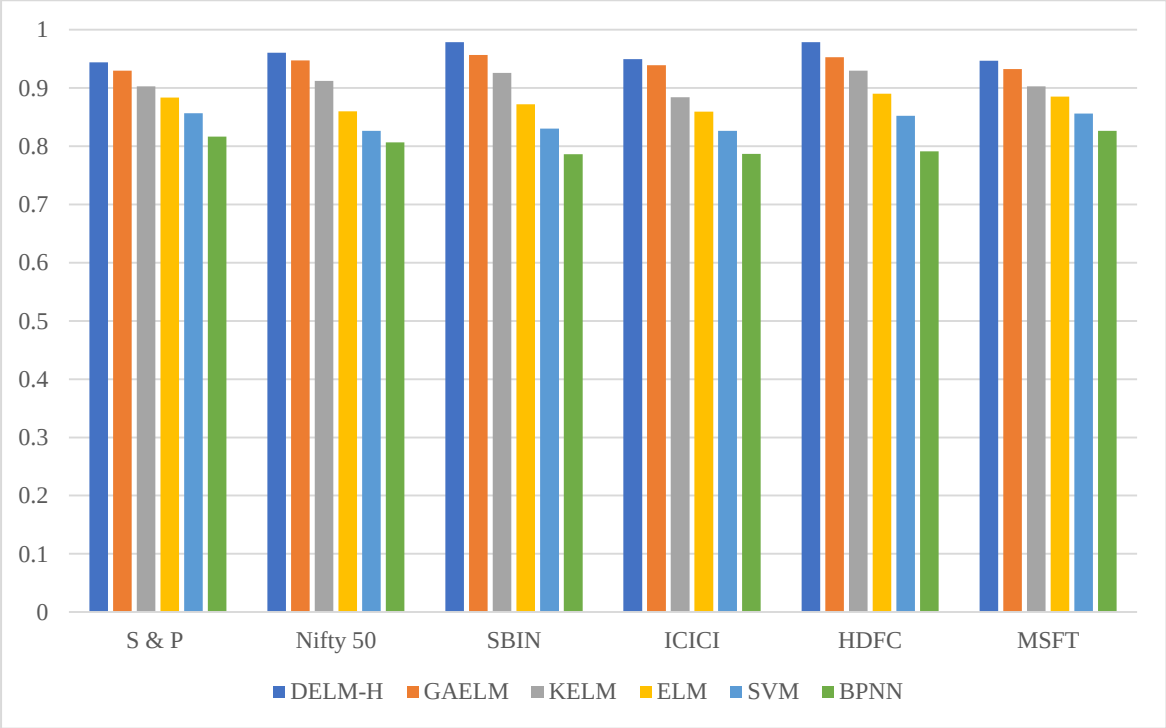

Figure 14*:* Testing performance comparisons for R-Square

4.7 Summary

Phase I of this research work discusses the ability of the developed DELM-H method. In comparison to previous related prediction algorithms, the recommended optimized extreme learning machine algorithm with deterministic weight adjustment technique using historical analysis produced low MSE, RMSE, MAE, MAPE values, and high R-Square values which is considerably efficient.