УДК 336.74

Comparative Analysis of Machine Learning Models for Money Demand Forecasting in the Indian Economy[1]

Shweta Sikhwal[1], Sreenjay Sen[2]

[1] National Research University Higher School of Economics,
18, Myasnitskaya Str., Moscow, 101000, Russian Federation.
E-mail: shsikkhval@hse.ru

[2] Upthink Edutech Private Limited,
Office No. 401, A Wing Lohia Jain IT Park, Paud Road, 411038 Pune, India.
E-mail: sreenjay07@gmail.com

The study investigates the predictive efficacy of various machine learning methodologies, encompassing Random Forest (RF) regression, Gradient Boosting (GB), Xtreme Gradient Boosting (XGBoost), Support Vector Regression (SVR), Least Absolute Shrinkage and Selection Operator (LASSO) regression, and a deep learning technique, specifically Long Short-Term Memory (LSTM). The benchmark method employed is the autoregressive (AR) model of order 1. With a focus on forecasting money demand for the Indian economy, a crucial component for achieving the Central Bank of India's inflation targeting objective, a comprehensive monthly dataset from 1997 to 2021 is utilized.

The obtained results underline the robust predictive capabilities of the employed models concerning both narrow and broad money demand forecasts. By employing a range of evaluation metrics, the study rigorously compares the predictive performance of these models. Using the expanding window cross validation with time series split, the models are cross-validated to ensure accurate forecasts

[1] Funding. This work was supported within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE).

Availability of data and material. Data used in this article is available upon request.

Acknowledgements. We are grateful to Ivan Stankevich for his valuable comments on the paper. We are also thankful to Marek Dabrowski and Svetlana Avdasheva for guiding us through the research. All mistakes and errors are entirely our own.

Shweta Sikhwal – Research Assistant at International Centre for the Study of Institutions and Development, HSE University.
Sreenjay Sen – Data Analyst at Upthink Edutech Private Limited.

The article was received: 29.08.2023/The article is accepted for publication: 08.02.2024.

of monetary aggregates. Moreover, the Diebold – Mariano test is utilized to evaluate and compare the quality of forecasts.

In particular, the research finds the superiority of LSTM and LASSO in predictive capabilities for narrow and broad money demand, respectively. These findings collectively contribute to enhancing the understanding of money demand prediction, thus facilitating informed decision-making within the realm of monetary policy.

Key words: narrow money demand; broad money demand; machine learning models; Random Forest regression; Gradient Boosting; Xtreme Gradient Boosting; Support Vector Regression; LASSO; long short-term memory.

JEL Classification: C53; E41; E47.

DOI: 10.17323/1813-8691-2024-28-1-133-158

For citation: Sikhwal S., Sen S. Comparative Analysis of Machine Learning Models for Money Demand Forecasting in the Indian Economy. *HSE Economic Journal.* 2024; 28(1): 133–158.

1. Introduction

In recent years, the rapid advancements in Machine Learning (ML) techniques have generated considerable interest in their application for accurate forecasting of key macroeconomic variables. These models offer promising opportunities for enhancing monetary policy decisions. When examining the stability of money demand, the Money Demand Function (MDF) serves as a vital tool, establishing a connection between money stocks and crucial macroeconomic indicators such as aggregate income and interest rates. A stable MDF plays a pivotal role in steering an economy towards its ultimate objective of price stability. According to [Laidler, 1982], the concept of a "stable demand for money function" implies the ability to explain variations in money holdings through functional relationships involving a concise set of variables. These relationships should produce statistically significant results within commonly accepted thresholds. With this objective in mind, our research focuses on exploring ML approaches to forecast the money demand of the Indian economy.

The demand for money function has garnered considerable empirical attention due to its profound implications for the effectiveness of monetary policy, seigniorage, inflation dynamics, and other crucial macroeconomic considerations. The new-monetarist economists emphasize the importance of money in the monetary policy framework. Thornton (2014) asserts that money plays a crucial role in monetary policy, primarily for its role in regulating the price level. Additionally, he argues that the perceived influence of the monetary authority in managing interest rates is overstated. Bordo, Jonung (2003) contend that economists and central bankers tend to view the money supply as directly proportional to the rate of inflation. Consequently, predictions of future inflation often hinge on a retrospective examination of money demand factors. King (2001) and Nelson (2003) caution against neglecting the importance of money, especially in the pursuit of maintaining price stability through monetary policy operations.

In the specific context of the Central Bank of India's current monetary policy framework, the accurate prediction of money demand assumes paramount importance for the successful implementation of policy measures. Since the formal adoption of inflation targeting in India in 2016, with a focus on maintaining price stability within a Consumer Price Index (CPI) band of 4% +/– 2%, the stability of money demand has emerged as a pivotal factor in achieving efficient price stability objectives. According to the quantity theory of money, the rate of inflation can be influenced by the growth rate of the money supply, assuming a stable money demand. Consequently, a comprehensive examination of the MDF and its precise forecasting becomes imperative. Understanding the factors that drive money demand and accurately predicting its behavior are essential for formulating effective monetary policy strategies. Therefore, our study aims to provide a thorough analysis of the MDF and employ advanced forecasting techniques to enhance the precision of money demand predictions.

Accurate estimation of the MDF is of importance for the effective implementation of monetary policy. The stability of the MDF refers to the degree to which the relationship between money demand and its determinants remains consistent over time. The MDF captures the influence of various factors, such as income, interest rates, and inflation, on the demand for money. Earlier studies have found a stable MDF after factoring in financial development [Arrau et al., 1995; Dekle, Pradhan, 1999; James, 2005; Adil et al., 2020]. If the MDF exhibits stability, it implies that the same set of determinants that currently affect money demand will continue to do so in the future. This stability enhances policymakers' ability to anticipate how changes in monetary policy or economic conditions will impact money demand and, consequently, the overall economy. By understanding and accurately estimating the stability of the MDF, policymakers can make informed decisions and formulate effective strategies to achieve desired monetary policy outcomes. In contrast, when the MDF exhibits instability, predicting the impact of economic changes on money demand becomes challenging, contributing to financial market volatility and broader economic instability. Policymakers are then faced with the task of revising policies more frequently and exercising greater caution in their decision-making processes to maintain economic stability.

The application of ML models in MDF forecasting provides valuable insights to policymakers, enabling a deeper understanding of these intricate dynamics. Predictions stemming from ML techniques may be more accurate than those derived from conventional approaches [Bajari et al., 2015]. By leveraging the predictive power of ML, our aim is to enhance the accuracy of MDF forecasts. Thus, the purpose of the study is to provide a better forecast of money demand for both narrow and broad monetary aggregates in India using ML and deep learning models and compare their efficacy.

Various methods have been proposed to understand money demand dynamics, including the Autoregressive Distributed Lag Model (ARDL), Error Correction Model (ECM), and Vector Autoregression (VAR). However, a deep learning approach, such as Long Short-Term Memory (LSTM), offers a more effective alternative due to its ability to capture non-linear relationships in time-series analysis. These models outperform traditional approaches and require less prior understanding of the complex relationships between variables. This highlights the potential of ML in enhancing money demand forecasting by uncovering hidden patterns and dynamics that traditional methods may overlook. In this paper, we employ an LSTM-based deep learning approach to forecast the MDF in the context of India. We also compare the performance of this approach with other models, including Random Forest (RF) regression, Gradient Boosting (GB), Xtreme Gradient Boosting (XGBoost), Support Vector Regression (SVR), Least Absolute Shrinkage and Selection Operator (LASSO) regression, and Autoregression (AR) of order 1.

The study is structured into six sections. Section 2 provides an extensive review of previous studies focusing on the stability of money demand. In Section 3, we describe the dataset and the methodology employed for our analysis. Section 4 describes model validation. The empirical analysis of the results is presented in Section 5. Finally, Section 6 concludes the study by summarizing the key findings.

2. Literature Review

The understanding of money demand behavior and its interaction with macroeconomic variables such as output and inflation has been a popular topic of research because of its importance in price stability. In this section, we will briefly discuss the empirical research done in this area.

Adil et al. (2020) examined money demand in India during the post-reform period using quarterly data from 1996 to 2016. Their findings, based on the co-integration approach and stability tests, indicated stable dynamics in the real money balances (M1 and M3) when incorporating financial innovation. Bahmani-Oskooee (1996) examined the stable long-run relationship between money demand and its determinants in the Japanese economy. Using quarterly data from 1975 to 1992, the study found a significant negative error correction term, indicating the Japanese economy's adjustment to short-run money market imbalances. The findings highlight the importance of considering income and interest rate fluctuations in monetary policy decision-making in Japan. Akinlo (2006) employed the ARDL approach in conjunction with the CUSUM and CUSUMSQ tests. They concluded a relatively stable relationship between M2 and key variables such as income, interest rate, and exchange rate.

Aggarwal (2016) analyzed India's MDF and observed temporary shocks in interest rates and M1, indicating the absence of a long-term equilibrium relationship. Bahmani-Oskooee et al. (2015) find evidence of a stable relationship between money demand and its covariates, indicating a stable long-run MDF. Barnett et al. (2022) examine the stability of money demand using Divisia measures. They find that Divisia monetary aggregates consistently outperform simple-sum measures, challenging the notion of money demand instability. The study emphasizes the importance of using accurate measures of money in economic analysis and has implications for monetary policy and understanding the money-economy relationship. Ball (2012) investigates the short-term dynamics of money demand, with a focus on the impact of interest rates, income, and inflation. The research examines the behavior of M1 in the U.S. from 1960 to 1993. The author points out the importance of understanding money demand, especially in the context of reversing the quantitative easing policy.

Goulet Coulombe et al. (2022) highlight the useful features of ML approaches over standard econometric models and study features such as non-linearity, regularization, and cross-validation used in ML. The authors conclude that ML is powerful in understanding the nonlinearity present in macroeconomic data, which is mostly present in periods of uncertainty, and incorporating this important feature would result in a better forecast. Gogas et al. (2019) use the SVR approach to investigate whether monetary aggregates can affect real economic activity in order to understand money neutrality. By employing a feed-forward artificial neural network (ANN), Pham et al. (2022) forecast the monthly inflation in Vietnam. They assert the proposed model to be reliable because of the narrow gap between actual and predicted inflation values. Nguyen et al. (2022) proposed the LSTM approach for macroeconomic forecasting on a total of 215 macroeconomic variables. The authors argue that the LSTM-based approach outperforms models such as

VAR in predicting a large number of macroeconomic variables simultaneously. Uyen et al. (2022) point out that even when OLS works well, LASSO regression can still be advantageous in variable selection and prediction.

Our research is closely aligned with the study conducted by [Ghose et al., 2021] as we delve into the realm of money demand forecasting and its predictive potential using ML models. In their work, they explore the effectiveness of ML models in comparison to traditional econometric approaches. While they found RF regression to be effective, we aim to further enhance the ML framework by employing deep learning approaches. Deep learning models, such as LSTM, offer the ability to learn complex patterns in the data through multiple processing layers.

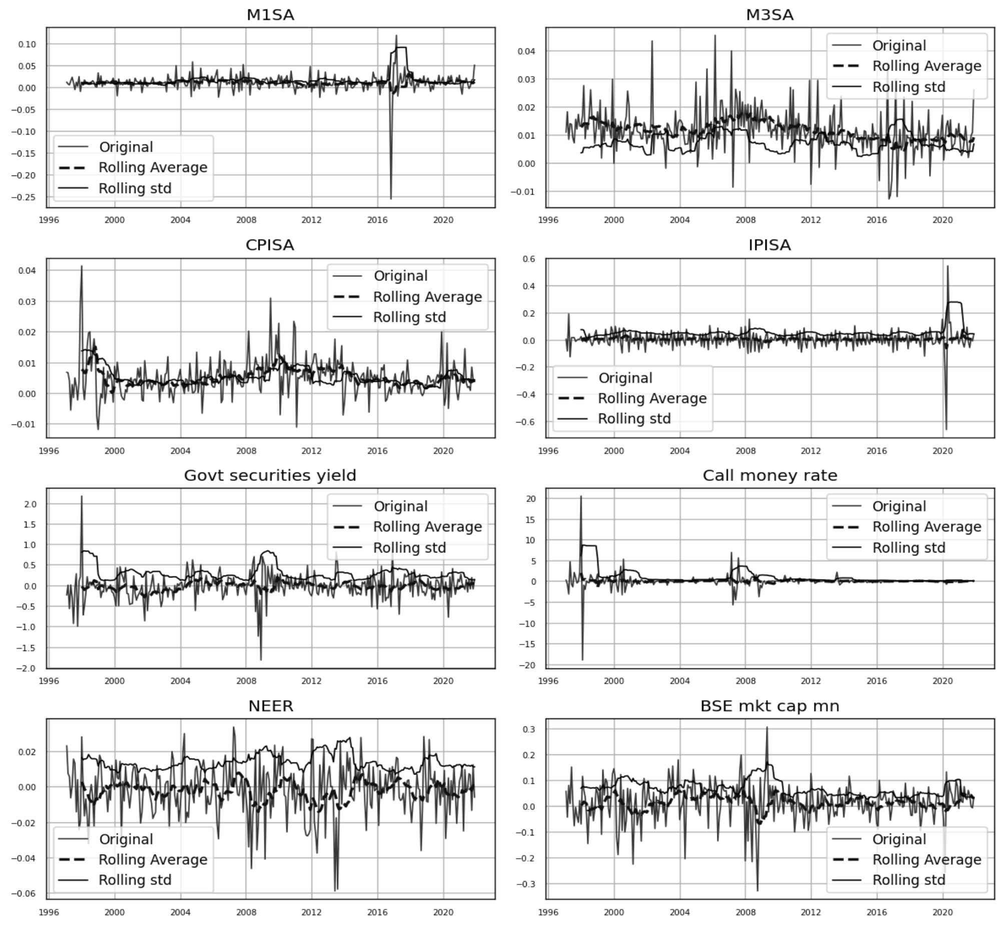

Fig. 1. Data series after first level differencing (Stationary dataset)

3. Dataset, Model Specification and Methodology

3.1. Dataset

The dataset used in our research comprises a monthly time series covering the period from 1997 to 2021, focusing on the Indian economy. The objective of our study is to forecast both the narrow monetary aggregate (M1) and the broad monetary aggregate (M3). To ensure accuracy in capturing the characteristics of these aggregates, we have to select an appropriate interest rate variable. Following the approach outlined by [Ball, 2012], we use the call money rate as the interest rate variable for the M1 model, while the government securities yield is employed for the M3 model. Income is approximated using the Index of Industrial Production (IIP), and the nominal effective exchange rate is incorporated to capture exchange rate dynamics. To gauge the financial stability of the Indian economy, we adopt the market capitalization of the Bombay Stock Exchange (BSE) as a proxy. The dataset is sourced from CEIC, and we apply seasonal adjustments to IIP, CPI, M1, and M3 using the X13 ARIMA seasonal adjustment approach.

3.2. Model Specification and Preliminary Analysis

The log linearized MDF is defined in the following:

$$\ln\left(\frac{M_{tj}}{P_t}\right) = \alpha_0 + \alpha_1 \ln Y_t + \alpha_2 R_{tj} + \alpha_3 \ln E_t + \alpha_4 \ln C_t + u_t. \tag{1}$$

In equation 1, $\ln$ represents the natural logarithm operator. M_{tj} denotes the nominal stock of money balances for time period t, where j represents either the narrow money balances (M1) or the broad money balances (M3). To calculate real money balances, we divide the nominal stock of money balances by the prevailing price level, represented by the CPI denoted as P_t. The variables in the equation include Y_t, which represents the IIP, R_t as the interest rate variable (call money rate for M1 and government securities rate for M3), E_t as the nominal effective exchange rate, and C_t as the market capitalization measured by the BSE stock valuation. We have taken a natural log of all the variables except for the interest rate variables, which are in percenttage form.

Before proceeding with the analysis, it is crucial to assess the stationarity of the time series. To determine stationarity, we conduct the Augmented Dickey – Fuller (ADF) and Phillips – Perron (PP) tests, considering a maximum lag length of 15. The results of the unit root test, presented in Section 5, indicate that all the data series are non-stationary at the level.

In our analysis, we employed the log difference transformation to address the non-stationarity of time series data and to model the continuous growth rates of economic variables. This approach involves taking the natural logarithm of each variable and then calculating the difference between consecutive logarithmic values. Figure 1 illustrates the transformed data.

3.3. Methodology

3.3.1. Autoregression of Order 1

The Autoregressive (AR) model of order 1, denoted as AR (1), models the next time step in a series as a linear function of the observation at the previous time step, plus a noise term. The AR (1) model is a single-variable (univariate) model that relies on the assumption that past values have a linear influence on future values. The AR (1) model can be formally expressed by the following equation:

$$Y_t = c + \phi_1 Y_{t-1} + \varepsilon_t. \quad (2)$$

Where Y_t is the value of the time series at time t; c is a constant term; ϕ_1 is the coefficient for the first lag of the series. It quantifies the influence of the previous time step's value (Y_{t-1}) on the current value Y_t. (Y_{t-1}) is the value of the time series at time $t-1$, the previous time step; ε_t is white noise error at time t, which is assumed to be a normally distributed random variable with mean zero and constant variance. This term accounts for randomness or unpredictability in the time series that is not explained by the lagged values. The parameter ϕ measures how changes in the previous time step affect the current value. A value close to 1 indicates a strong positive relationship with the previous step, a value close to –1 indicates a strong negative relationship, and a value close to 0 indicates a weak relationship.

3.3.2. Random Forest Regression

The Random Forest (RF) method is an extension of the decision tree technique, which employs a flowchart-like tree structure. The flowchart-like structure of decision trees aids in decision-making by visualizing the paths taken to reach a conclusion based on input features. At each internal node, a decision is made by evaluating an attribute against a threshold. Branches from these nodes lead to new questions or decisions, culminating in leaf nodes that represent the final outcomes or predictions. This structure simplifies complex decision processes, enabling both straightforward interpretation of how decisions are made and easy identification of the most influential factors in the decision-making process. The learning process of a decision tree involves dividing the source set into subsets based on attribute value tests, recursively partitioning the derived subsets. This recursion continues until all nodes within a subset have the same target variable value, or when further splitting does not contribute significantly to predictions.

It builds multiple decision trees and merges them together to get a more accurate and stable prediction. Each decision tree in a random forest is trained on a bootstrapped sample of the data, meaning that for each tree, a random sample of the training dataset is selected with replacement. This process introduces diversity among the trees, which helps in reducing the variance of the model. When growing each tree, at each split, Random Forest randomly selects a subset of features rather than using all features. This randomness helps in making the model more robust and less prone to overfitting on the training data. The base learners in Random Forest are decision trees. Each tree is grown to its maximum length without pruning, allowing it to capture complex patterns in the data. However, individual trees may overfit to their bootstrapped sample. The ensemble approach mitigates this overfitting.

The following step includes aggregating the predictions. Here, for regression tasks, Random Forest predicts the output based on the average predictions of all the individual trees. Mathematically, if there are N trees, the final prediction $\hat{y}$ is calculated as:

(3) $$\hat{y} = \frac{1}{N}\sum_{i=1}^{N} T_i(X).$$

In equation 3, $T_i(X)$ represents the prediction made by the i^{th} decision tree for an input X. The ensemble's final prediction, $\hat{y}$, is the average of all individual tree predictions, effectively combining their insights to improve accuracy and stability. This averaging process helps in reducing the variance of the predictions, making the model more stable and accurate on unseen data. During training, N decision trees are constructed using different bootstrap samples of the training data. Each tree is built considering a random subset of features at each split. N controls the number of trees in the forest. More trees can increase accuracy but also computational cost. For a new input, each of the N trees gives a prediction. In regression, the final output is the average of these N predictions.

The model operates by creating an ensemble of diverse trees, each trained on a random subset of the data with replacement (bootstrap sample), and using a random subset of features for splitting nodes. The final prediction is typically the average of all tree predictions. The RF algorithm generates numerous decision trees from randomly sampled databases. For each subset of the randomly sampled data, a decision tree is constructed following the process described above. However, a key distinction lies in the selection of input variables for each tree, which is done randomly. This random selection reduces the correlation between trees in the forest, mitigating the risk of overfitting. The RF algorithm combines the predictions from multiple decision trees trained on bootstrapped and randomly selected subsets of the training data, leading to improved robustness and generalization compared to a single decision tree regression.

3.3.3. Gradient Boosting

Gradient Boosting (GB) is an ensemble learning method that combines multiple individual models to create a robust and accurate predictor. The core idea of GB revolves around iteratively building an ensemble of simple models, often referred to as weak learners or base models, and intelligently combining them to minimize overall prediction error.

It employs the process of assembling multiple weak learners to create powerful models. The method operates through iterative steps, where each new model is trained to reduce the loss function of the previous ensemble, such as mean squared error or cross-entropy, by utilizing gradient descent. In each iteration, the algorithm computes the gradient of the loss function with respect to the current ensemble's predictions and constructs a new weak model to minimize this gradient. The predictions of the new model are then integrated into the ensemble, and this process continues until a specified stopping criterion is met.

The GB model starts with a base model, F_0, which could be the mean of the target values for regression tasks. It provides a starting point for the iterative process. For each iteration m, a new decision tree is added to the ensemble. It calculates the gradient of the loss function

$L(y,F)$ with respect to the predictions from the current model, F_{m-1}, for each observation in the dataset. This gradient indicates the direction in which the model's predictions need to be adjusted to reduce the loss. A weak learner (usually a decision tree) is then fitted to these gradients. Specifically, for regression, the tree is fitted to the negative gradients (pseudo-residuals) of the loss function with respect to the predictions. Weak Learner Fitting in GB involves training simple models on the residuals or errors of the ensemble's predictions from the previous step. These weak learners are usually decision trees with a limited depth, allowing them to capture only a portion of the data's variance. It is done to sequentially improve the ensemble by adding models that address the most significant current errors, making the overall prediction more accurate with each addition. After fitting, the predictions from the new tree are combined with the predictions from the existing ensemble to update the model. This process is controlled by a learning rate parameter to prevent overfitting. The framework of GB can be explained as the following.

The initial model F_0 is typically a constant value:

$$F_0(x) = \arg\min \sum\nolimits_{i=1}^{n} L(y_i, \gamma). \tag{4}$$

For regression, γ can be the mean of the target values y.

Then, at each step t, the residuals r_{it} for each observation i are calculated as the negative gradient of the loss function L with respect to the prediction:

$$r_{it} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{t-1}(x)}. \tag{5}$$

A decision tree $h_t(x)$ is then fitted to the residuals r_{it} from the previous step. This tree attempts to correct the errors made by the ensemble so far. The next step is to find the optimal step size γ_t for each leaf of the tree h_t, which minimizes the loss when added to the current model:

$$\gamma_t = \arg\min_{\gamma} \sum_{i \in I_j} L(y_i, F_{t-1}(x_i) + \gamma h_t(x_i)), \tag{6}$$

where I_j is the set of indices of samples ending up in leaf j.

Now, the output of the weak learner is then scaled by a factor, η, called the learning rate, and added to the current model's prediction to update it:

$$F_t(x) = F_{t-1}(x) + \eta \sum\nolimits_j \gamma_{tj} h_t(x). \tag{7}$$

Here, γ_{tj} is the optimal step size for leaf j and $h_t(x)$ is the prediction of the t^{th} tree.

These steps are repeated for a specified number of iterations or until additional trees do not significantly reduce the loss function.

3.3.4. Xtreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is based on the gradient boosting framework, which iteratively combines multiple weak learners (typically decision trees) to create a strong ensemble model. Each new tree is added to correct the residuals or errors made by the previous ensemble of trees. XGBoost improves upon the traditional gradient boosting method by incorporating regularization terms in its objective function and optimizing both the model's performance and computational efficiency.

XGBoost constructs a new weak learner, a decision tree, specifically to predict the residuals. This new model focuses on learning from the mistakes made by the initial model, aiming to improve its predictive capabilities. It uses a gradient descent optimization technique to minimize the loss function, such as mean squared error or cross-entropy, by finding the best splits for the decision tree nodes. The iterative process continues, with XGBoost adding new weak learners to the ensemble in each iteration. At each step, the algorithm computes the gradients of the loss function with respect to the current ensemble's predictions. It then constructs a new weak model to minimize these gradients, effectively addressing the errors or residuals from the previous model.

As the iterations progress, the ensemble of weak learners gradually improves its predictive performance, capturing complex relationships and patterns in the data. The predictions of the new models are combined with those of the existing ensemble using a carefully chosen learning rate. This learning rate controls the contribution of each model to the final prediction, ensuring a balanced and well-optimized ensemble.

The XGBoost process can be defined as follows. By incorporating the regularization terms in its objective function and optimizing both the model's performance and computational efficiency. The objective function in XGBoost combines a loss function L and a regularization term Ω, which is applied to each tree in the ensemble. The overall objective to be minimized at each step (for each tree added) can be written as:

$$obj = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f)_k, \tag{8}$$

where n is the number of training samples; y_i is the actual value of the i^{th} sample; $\hat{y}_i$ is the predicted value for the i^{th} sample, which is the sum of the predictions from all K trees up to the current tree. f_k represents an individual tree in the ensemble. L is the loss function that measures the difference between the actual and predicted values. Ω is the regularization term for the trees, which is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{9}$$

Here, T is the number of leaves in the tree; w_j is the weight assigned to the j^{th} leaf; γ and λ are parameters that control the complexity of the model, with γ penalizing the number of leaves and λ penalizing the magnitude of the leaf weights.

XGBoost uses a second-order approximation to optimize the objective function. For each tree, it calculates the gradient (g_i) and the Hessian (h_i) of the loss function with respect to the prediction for each training instance. These are used to find the optimal structure of the tree and the best leaf weights. The gradient and Hessian for a given instance are:

$$g_i = \partial_{\hat{y}_i} L(y_i, \hat{y}_i), \quad h_i = \partial^2_{\hat{y}_i} L(y_i, \hat{y}_i). \tag{10}$$

When building each tree, XGBoost selects splits that maximize the gain in the objective, which is derived from the gradient and Hessian information. The gain of a split is given by:

$$Gain = \frac{1}{2}\left(\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\left(\sum_{i \in I_L} h_i + \lambda\right)} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\left(\sum_{i \in I_R} h_i + \lambda\right)} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\left(\sum_{i \in I} h_i + \lambda\right)} \right) - \gamma. \tag{11}$$

Where I_L and I_R are the sets of instance indices in the left and right child regions of the split, respectively; I is the set of instance indices for the parent region. The sums of gradients $\left(\sum g_i\right)$ and Hessians $\left(\sum h_i\right)$ are calculated over the instances in the respective sets. This formula takes into account the reduction in loss due to the split (first three terms) and subtracts the regularization penalty for adding a new leaf (γ).

After determining the structure of a tree, XGBoost calculates the optimal weight for each leaf to minimize the objective function. The optimal weight for a given leaf is:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{12}$$

where I_j is the set of instance indices in the j^{th} leaf.

3.3.5. Support Vector Regression

Support Vector Regression (SVR) is a type of Support Vector Machine (SVM) used for regression tasks. SVR applies the principles of machine learning to predict continuous values. SVR finds a function that has at most ϵ deviation from the actual target values for all the training data, and at the same time, is as flat as possible. Here flatness means seeking a function that does not oscillate too much, which typically translates into finding a model with a small coefficient in a high-dimensional space.

SVR is considered a nonparametric technique because it does not rely on a fixed functional form or assume any specific underlying distribution. Instead, SVR uses kernel functions to map the data into a higher-dimensional feature space, where it can find linear relationships and patterns that might not be evident in the original input space.

It finds a function $f(x)$ that approximates the true relationship between the predictor variables (denoted as x) and the observed response values (denoted as y_n) from the training data. The goal is to ensure that the difference between the predicted values and the true values (the residuals) is no greater than a specified margin or tolerance called ϵ. This margin is an important parameter in SVR as it determines the trade-off between model complexity and accuracy. SVR aims to minimize the empirical risk, which is the sum of the ϵ-insensitive loss function and a regularization term. The ϵ-insensitive loss function penalizes the model for large deviations between the predicted and actual response values while allowing small deviations within the margin (ϵ). The loss function can be written as:

$$\text{(13)} \qquad \begin{cases} 0 & if\,|y-f(x)|\le\epsilon \\ |y-f(x)|-\epsilon & otherwise \end{cases}.$$

Where y is the actual value, $f(x)$ is the predicted value, and ϵ is the margin of tolerance.

It uses kernel function to transform the original feature space into a higher-dimensional space where a linear separator is sought. It minimizes the following objective function, which represents a trade-off between the flatness of $f(x)$ and the amount by which predictions fall outside the ϵ-insensitive zone:

$$\text{(14)} \qquad \min_{w,b,\xi,\xi^*}\frac{1}{2}\|w\|^2+C\sum\nolimits_{i=1}^{n}\left(\xi_i+\xi_i^*\right),$$

subject to the constraints
$$\begin{aligned} & y_i-\langle w,x_i\rangle-b\le\epsilon+\xi_i, \\ & \langle w,x_i\rangle+b-y_i\le\epsilon+\xi_i^*, \\ & \xi_i,\xi_i^*\ge 0,\forall i. \end{aligned}$$

Here, w is the weight vector, b is the bias term, C is the regularization parameter, ξ_i and ξ_i^* are slack variables that measure the degree of misfit for data points outside the ϵ margin. The optimization problem is often solved in its dual form to facilitate the use of kernel functions, allowing the algorithm to operate in the high-dimensional feature space without explicitly computing the coordinates of the data in that space. The dual problem involves Lagrange multipliers and maximizes a Lagrangian function derived from the primal problem. Once the optimal w and b are found, the regression function can be used for prediction:

$$\text{(15)} \qquad f(x)=\langle w,x\rangle+b.$$

3.3.6. Least Absolute Shrinkage and Selection Operator

LASSO (Least Absolute Shrinkage and Selection Operator) regression is a type of linear regression that includes a regularization term. The regularization term is the $L1$ norm of the coefficients, which encourages sparsity in the coefficients. In other words, it can reduce the coefficients of less important features to zero, effectively performing feature selection as part of the regression. This characteristic makes LASSO particularly useful for models with a large number of features, some of which might be irrelevant to the prediction.

LASSO seeks to find the best-fitting model while simultaneously imposing a constraint on the sum of the absolute values of the coefficients. This constraint acts as a regularization term and encourages the coefficients of less relevant variables to be set to exactly zero. As a result, LASSO performs variable selection by effectively shrinking less important predictors to eliminate their contribution to the model. The regulation parameter, λ, controls the trade-off between fitting the data well and keeping the model coefficients sparse. $\lambda = 0$ corresponds to ordinary least squares regression, while very large values of λ can lead to all coefficients being shrunk to zero. The mathematical formulation of LASSO involves minimizing the residual sum of squares, similar to the ordinary least squares method, subject to a constraint on the $L1$-norm (sum of absolute values) of the coefficient vector. This constraint controls the amount of regularization applied to the model. This can be formulated as the following optimization problem:

$$\min_{\beta}\left\{\frac{1}{2n}\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\left|\beta_j\right|\right\}. \tag{16}$$

Where, y_i is the observed response for the i^{th} observation; X_{ij} is the value of the j^{th} predictor for the i^{th} observation; β_j are the coefficients to be estimated; β_0 is the intercept term; n is the number of observations; p is the number of predictors. λ is the regularization parameter that controls the degree of shrinkage applied to the coefficients. LASSO can set some coefficients to zero, effectively selecting a simpler model that relies on fewer features. This is particularly beneficial in scenarios with high-dimensional data or when the goal is to identify a subset of relevant predictors. As λ varies, the solution path of the LASSO coefficients can be plotted, showing how each coefficient enters or leaves the model as λ changes. This path provides insights into the relative importance of the features. The higher the value of λ, the greater the amount of shrinkage, leading to more coefficients being set to zero.

3.3.7. Long Short-Term Memory

The Long Short-Term Memory (LSTM) model is well-suited for capturing both short-term and long-term dependencies within the data due to its ability to incorporate recurrent connections. These connections enable the model to retain information across different time steps, facilitating the understanding of complex patterns and features in the sequential data. LSTM networks process input data sequentially, one time stamp at a time. During each time stamp, the network takes into account the current input, combines it with the previous memory state, and performs various computations using the gates and activation functions. This iterative process allows LSTMs to learn and update their internal state, effectively capturing both short-term and long-term dependencies within the data.

Figure 2 illustrates the architecture of an LSTM cell, which is the building block of the LSTM model. At each time stamp t, the LSTM cell takes an input vector x_t and produces an output vector

h_t. Additionally, it receives the previous output value (h_{t-1}) and cell state value (C_{t-1}) from the previous time step. The LSTM cell consists of three gates that regulate the flow of information: the input gate (i_t), the forget gate (f_t), and the output gate (o_t). Each gate utilizes a sigmoid activation function, producing a vector of values ranging from 0 to 1 that determines the amount of information to retain or forget.

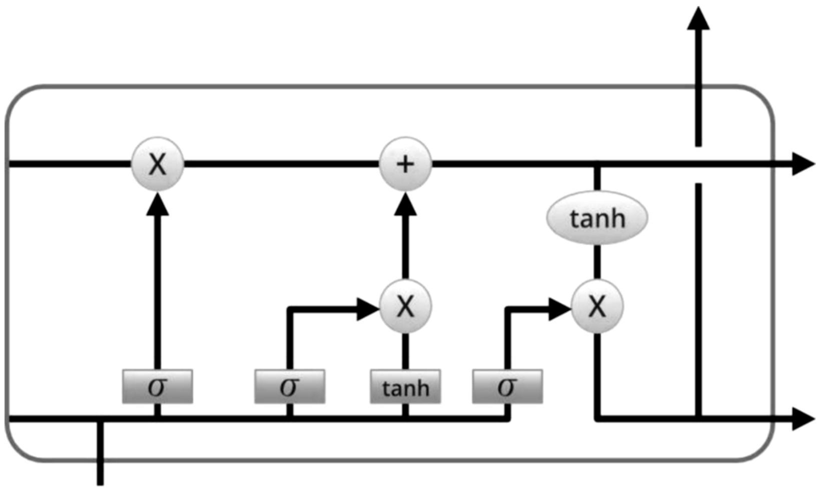

Fig. 2. Structure of an LSTM cell (Sourced from Picture from Christopher Olah's blog)

The input gate (i_t) controls the information to be stored in the memory cell (c_t). It takes the previous hidden state (h_{t-1}) and the current input (x_t) as inputs. The forget gate (f_t) determines the extent to which the previous memory cell content is retained or forgotten. It also takes the previous hidden state and current input as inputs. The output gate (o_t) determines the portion of the memory cell content that should be exposed as the output.

Mathematically, LSTM can be described as:

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right), \tag{17}$$

where i_t is the input gate vector, $\sigma(\cdot)$ represents the sigmoid function, W_i is the weight matrix and b_i is the bias vector for the input gate.

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right), \tag{18}$$

where f_t is the forget gate vector, $\sigma(\cdot)$ represents the sigmoid function, W_f is the weight matrix and b_f is the bias vector for the forget gate.

Additionally, a memory cell stores the information over time by selectively adding or removing information through the input and forget gates. It uses a hyperbolic tangent activation function to regulate the values that flow through it.

(19) $$\tilde{c}_t = \tanh\left(W_c\left[h_{t-1}, x_t\right] + b_c\right),$$

where $\tilde{c}_t$ is the candidate memory cell content which represents the new information that can be stored in the memory cell, $\tanh(\cdot)$ is a hyperbolic tangent activation function to regulate the values that flow through it, W_c and b_c are the weight and bias parameters.

(20) $$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t,$$

where c_t is the memory cell content which is updated based on the input gate $\left(i_t\right)$, forget gate $\left(f_t\right)$, and candidate memory cell content $\left(\tilde{c}_t\right)$. The forget gate controls how much of the previous memory cell content $\left(c_{t-1}\right)$ is retained, while the input gate determines how much of the candidate memory cell content $\left(\tilde{c}_t\right)$ is stored.

(21) $$o_t = \sigma\left(W_0\left[h_{t-1}, x_t\right] + b_0\right),$$

where o_t is output gate vector, $\sigma(\cdot)$ represents the sigmoid function, W_0 is the weight matrix and b_0 is the bias vector for the output gate. It calculates a new hidden state h_t as following:

(22) $$h_t = o_t \cdot \tanh\left(c_t\right).$$

By iteratively updating the cell state and hidden state based on the input, forget, and output gates, the LSTM model effectively captures the complex dynamics and dependencies within the time series data. This enables the model to generate accurate forecasts by learning from the sequential patterns present in the dataset.

4. Model Validation

4.1. Expanding Window Cross-validation with Time-Series Split

To ensure our model evaluations are both accurate and robust, we've structured our dataset into training and testing segments. Our method combines time-series splitting with expanding window cross-validation, specifically designed for the sequential nature of time-series data. This strategy mimics real-world conditions where models learn from an ever-increasing dataset, validating on future data points to guarantee temporal integrity.

This integrated methodological framework begins by dividing the dataset into an initial training segment, constituting 80% of the data, and a testing segment, making up the remaining 20%. The expanding window cross-validation works in an iterative process, where the training dataset starts with a small subset and is incrementally expanded by including subsequent data points in each fold. In our study, these folds are represented by K , where K takes a value from 2 to 7. This ensures our model progressively learns from a growing historical dataset, reflecting a natural accumulation of data over time. Unlike traditional cross-validation techniques, which may disregard the temporal order by randomly partitioning data, our method maintains the sequential progression of time by ensuring that the model is always trained on data preceding the test set. This is crucial for time series forecasting, where the validity of predictions heavily relies on the historical context and temporal dynamics of the dataset.

Moreover, by combining time-series split with the expanding window technique, we enhance the robustness of our model evaluation. This combination not only facilitates a thorough assessment of the model's performance over various stages of the dataset but also allows for a more nuanced understanding of how the model adapts to and predicts based on an expanding body of data over time. The expanding training set mimics a realistic setting where forecasts are based on an ever-growing historical context, offering insights into the model's scalability and adaptability to new data. This comprehensive evaluation framework thereby ensures a rigorous validation process, which is indispensable for developing reliable and accurate forecasting models in the realm of time series analysis.

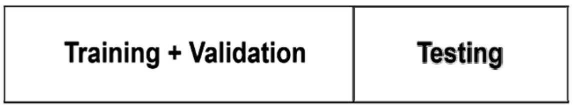

Fig. 3. Cross validation

4.2. Hyperparameter Tuning

To further optimize the performance of the models, we employ hyperparameter tuning. Hyperparameters are parameters that are not learned from the data but are set by the user before training the model. They significantly impact the model's performance and generalization ability.

For RF, GB, XGBoost, SVR, and LASSO, we have used the default hyperparameters that come with the "sklearn" library.

For LSTM, we have "pytorch" library and have selected from a range of hyperparameters (hidden size, number of layers, dropout rate, and learning rate). In an exhaustive grid search method[2], we explore all possible combinations of hyperparameter values, enabling us to find the best set of hyperparameters that yield the highest model performance. We use nested loops to iterate over different values of hyperparameters. The hyperparameters being tuned are the hidden size, number of layers, dropout rate, and learning rate. These hyperparameters control the architecture and training of the LSTM model. Inside the nested loops, the LSTM model is trained on the training data using the specified hyperparameters. The best combination of hyperparameters for each split is determined by the lowest validation loss using the MSE criterion. Note that we have run the LSTM for 300 epochs for both the M1 and M3.

[2] The table containing the list of the best hyperparameters for each fold (represented by K) is provided in the appendix section.

4.3. Evaluation Metrics

To evaluate the models, we employ several metrics, namely Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percent Error (MAPE), Symmetric Mean Absolute Percentage Error (SMAPE), and Theil inequality coefficient (TIC). These metrics provide valuable insights into the performance of our forecasting models. The employed metrics are calculated as follows:

The Mean Squared Error (MSE) measures the average of the squared differences between the original and predicted values in the dataset. It quantifies the variance of the residuals. The formula for MSE is:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(y_t - \hat{y}_t\right)^2. \tag{23}$$

The Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error. It represents the standard deviation of the residuals. The formula for RMSE is:

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y_t - \hat{y}_t\right)^2}. \tag{24}$$

The Mean Absolute Percent Error (MAPE) measures the average absolute percentage difference between the actual and predicted values. It provides a percentage measure of the average forecasting error. The formula for MAPE is:

$$MAPE = 100 \cdot \frac{1}{N}\sum_{t=1}^{N}\left|\frac{\left(y_t - \hat{y}_t\right)}{y_t}\right|. \tag{25}$$

The Symmetric Mean Absolute Percentage Error (SMAPE) is a variation of MAPE that addresses the issue of asymmetric errors. It calculates the average absolute percentage difference between the actual and predicted values, taking into account the magnitudes of both. The formula for SMAPE is:

$$SMAPE = 100 \cdot \frac{2}{N}\sum_{t=1}^{N}\frac{\left|y_t - \hat{y}_t\right|}{\left|y_t\right| + \left|\hat{y}_t\right|}. \tag{26}$$

The Theil inequality coefficient (TIC) evaluates the forecasting performance by comparing the forecasted and actual values relative to the average size of the variables. It considers both the forecasted and actual values in relation to their magnitudes. The formula for TIC is:

$$TIC = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y_t - \hat{y}_t\right)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N} y_t^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N} \hat{y}_t^2}}. \tag{27}$$

In these formulas, y_t represents the actual value at time period t, and $\hat{y}_t$ represents the predicted value at time period t. N denotes the total number of observations, which in our case is 48.

4.4. Forecasting Accuracy

We use the Diebold-Mariano (DM) test to check the forecasting accuracy of the employed models. The DM test is a statistical hypothesis test used to compare the forecasting accuracy of two or more forecasting models. It helps determine whether one forecasting model is significantly better than another in terms of predictive performance. Earlier, we used several metrics that signify the forecast accuracy based on the error term, where a forecast is considered better with a low error measure. However, we also need to understand if the difference in these error terms is significant enough to consider one forecasted result better than the other. The test assesses whether one model's forecasts are significantly better than the others, based on the Mean Squared Error (MSE) criterion.

We employ the DM test by incorporating the Harvey adjustment. It accounts for potential autocorrelation and heteroskedasticity in the forecast errors. The original DM test assumes that the forecast errors are independent and identically distributed (i.i.d.), which might not be the case in some real-world scenarios. The Harvey adjustment addresses this issue by estimating and accounting for the correlation among the forecast errors over time. It adds a correction factor to the DM test statistic to make it more robust to autocorrelation and heteroskedasticity. The adjustment considers the lagged covariance of the forecast errors and adjusts the standard errors of the test statistic accordingly. This adjustment provides more accurate results when comparing the predictive performance of different models.

5. Empirical Analysis

To ensure an accurate prediction of money demand, it is crucial to confirm the stationarity of the variables. We employ two commonly used unit root tests, namely the Augmented Dickey-Fuller (ADF) test and the Phillips – Perron (PP) test, to examine the presence of unit roots, which indicates if the variable is non-stationary. The null hypothesis for both tests is that the variable has a unit root, implying non-stationarity. In Table 1, we observe that the unit root null hypothesis cannot be rejected for the series at the level, indicating non-stationarity. To address this, we take the first difference of all the variables, transforming them into a first-order integrated I (1) series. Subsequent unit root tests confirm that the differenced series are stationary at the 1% level of significance.

With stationary variables, we proceed to compare the forecasting accuracy of different models, including AR (1), RFR, GB, XGBoost, SVR, LASSO, and LSTM. The dataset is divided into training and testing sets, and to avoid overfitting on the testing set, we employ expanding window cross-validation with k-fold iterations (K = 2 to 7). This approach partitions the training set into different folds, allowing us to train the model on one part and validate it on the remaining part. This helps improve the accuracy of the model when applied to the testing set.

Table 1.

Unit root test: ADF statistic and Phillip – Perron statistic

Variable	ADF statistic				PP Statistic			
	Level	P value	First difference	P value	Level	P value	First difference	P value
M1	–0.533	0.885	–8.102	0.000	–0.358	0.917	–17.163	0.000
M3	–2.115	0.2385	–2.154	0.223	–3.643	0.005	–20.414	0.000
CPI	0.285	0.977	–4.431	0.000	–0.323	0.922	–13.758	0.000
IPI	–1.811	0.375	–14.149	0.000	–2.067	0.258	–35.215	0.000
Govt securities yield	–2.476	0.121	–8.124	0.000	–2.829	0.054	–18.120	0.000
Call money rate	–5.559	1.559	–9.929	0.000	–9.675	0.000	–39.363	0.000
NEER	–0.629	0.864	–14.365	0.000	–0.373	0.914	–14.161	0.000
BSE mkt cap	–0.482	0.895	–15.66	0.000	–0.516	0.889	–15.612	0.000

Table 2.

Prediction of M1

	K = 6				
	MSE	RMSE	MAPE	SMAPE	TIC
AR (1)	0.00406	0.06373	0.43964	0.44097	0.00260
RF	0.00707	0.08409	0.58642	0.58874	0.00343
GB	0.00963	0.09813	0.68041	0.68357	0.00401
XGBoost	0.00720	0.08488	0.59384	0.59618	0.00346
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	0.00617	0.07854	0.518799	0.51681	0.00318

To evaluate the forecasting accuracy of the models, we analyze the errors, which are the differences between the true values and the predicted values. Several criteria, including MSE, RMSE, MAPE, SMAPE, and TIC, are used to assess the performance of the models. Table 2 and 3 present the forecasted results for M1 and M3 over K folds, where k takes the value 6[3]. The AR (1) model

[3] The table containing the results for K = 2 to 7 folds is presented in the appendix section.

serves as a benchmark. Considering the low values of the evaluation metrics in the evaluated results, it suggests that all models have a certain level of efficacy in forecasting the employed models. However, it can be important to understand the accuracy of each model by comparing their predictive performance. The analysis of prediction models for M1 money supply reveals distinct performance levels across various methodologies. SVR demonstrates the lowest error metrics (MSE, RMSE, MAPE, SMAPE), suggesting its superior ability to forecast with precision. On the other hand, LASSO and GB models exhibit relatively higher error rates, indicating potential overfitting or underfitting issues, thus proving less effective for this dataset. LSTM's balanced error metrics suggest its capability in handling complex temporal patterns, making it a viable option for time series forecasting.

The performance metrics for forecasting M3 money supply, evaluated across different models, showcase varied efficacy and accuracy. The LSTM model stands out with the lowest MSE, RMSE, MAPE, SMAPE, and TIC values, indicating it is the most accurate and efficient model for this dataset. This suggests LSTM's superior ability in capturing the complex temporal dependencies inherent in the M3 money supply data. GB and RF models also exhibit strong performance, with very competitive error metrics, signifying their effectiveness in handling time series forecasting for M3, albeit slightly less accurately than LSTM. These insights highlight the importance of model selection based on the specific characteristics of the financial time series being forecasted, with LSTM accurately modeling the M3 money supply.

Table 3.

Prediction of M3

	K = 6				
	MSE	RMSE	MAPE	SMAPE	TIC
AR (1)	0.00428	0.06541	0.38088	0.37976	0.00238
RF	0.00334	0.05775	0.32549	0.32462	0.00210
GB	0.00329	0.05738	0.32292	0.32206	0.00208
XGBoost	0.00560	0.07482	0.45037	0.44891	0.00272
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.00254	0.05042	0.28840	0.28774	0.00183

To provide a more conclusive assessment of forecasting accuracy, we employ the DM test. This test allows us to compare the quality of forecasts and determine the forecasting accuracy of the models. The results of the DM test are presented in Tables 4 and 5 for M1 and M3, respectively. The matrix format of the results compares the forecasts of two models in each cell. The DM test is employed for K = 6. According to the DM test, for the M1 forecast, the LSTM model's forecasts are statistically significantly different from the other models, indicating the better forecasting performance of LSTM. When comparing the forecasts made by LSTM, the difference is statistically significant at the 1% level for all of the models. Whereas, for M3, LASSO is performing better than other employed models.

Table 4.

Diebold – Mariano test for M1

K = 6	AR (1)	RF	GB	XGBoost	SVR	LASSO	LSTM
AR (1)	–	–8.504***	–4.063***	–1.206	2.665**	–7.616***	–7.724***
RF	8.504***	–	7.503***	6.670***	–11.996***	–7.436***	–7.608***
GB	4.063***	–7.503***	–	4.872***	4.882***	–7.476***	–7.632***
XGBoost	1.206	–6.670***	–4.872***	–	3.529***	–7.467***	–7.672***
SVR	–2.665**	11.996***	–4.882***	–3.529***	–	–7.472***	–7.788***
LASSO	7.616***	7.436***	7.476***	7.467***	7.472***	–	–7.133***
LSTM	**7.724*****	**7.608*****	**7.632*****	**7.672*****	**7.788*****	**7.133*****	**–**

Note: *, ** and *** are 10%, 5%, and 1% level of significance.

Table 5.

Diebold – Mariano test for M3

K = 6	AR (1)	RF	GB	XGBoost	SVR	LASSO	LSTM
AR (1)	–	6.257***	6.409***	–6.783***	–6.636***	–9.215***	8.331***
RF	–6.257***	–	–2.544**	–6.887***	–6.548***	–9.164***	2.924***
GB	–6.409***	2.544**	–	–7.532***	–7.882***	–9.417***	4.881***
XGBoost	6.783***	6.887***	7.532***	–	6.746***	–9.749***	7.265***
SVR	6.636***	6.548***	7.882***	–6.746***	–	–9.323***	7.746***
LASSO	**9.215*****	**9.164*****	**9.417*****	**9.749*****	**9.323*****	**–**	**9.245*****
LSTM	–8.331***	–2.924***	–4.881***	–7.265***	–7.746***	–9.245***	–

Note: *, ** and *** are 10%, 5%, and 1% level of significance.

6. Conclusion

The stability of the Money Demand Function (MDF) plays a pivotal role in the effective modeling and analytical examination of how monetary and fiscal policies influence economic outcomes. This aspect gains heightened importance in the backdrop of India's central bank adopting an inflation-targeting framework. Within this framework, the precise forecasting of a stable MDF is indispensable for the successful implementation of policies aimed at controlling inflation. The Quantity Theory of Money emphasizes the significance of a stable MDF in guiding inflation-targeting efforts. This theory posits that the growth of the money supply determines the long-term inflation rate, and a stable MDF is a key component in this relationship.

The field of Machine Learning (ML) has seen significant advancements, opening up novel opportunities for the forecasting of economic indicators with enhanced precision. Notably, the Long Short-term Memory (LSTM) model, which is rooted in deep learning techniques, has demonstrated remarkable proficiency in learning from complex and nonlinear data patterns to make accurate predictions. Our research attempts to explore the capabilities of LSTM models extensively, assessing their performance in forecasting money demand compared to other ML models, including Random Forest (RF), Gradient Boosting (GB), XGBoost, Support Vector Regression (SVR), LASSO, and Autoregressive (AR) models.

To assess the forecasting accuracy of these diverse models, we employ a comprehensive set of evaluation criteria, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), Symmetric Mean Absolute Percentage Error (SMAPE), and Total Information Criterion (TIC). Lower values across these metrics are indicative of superior model performance and a closer fit to the actual dataset. Our empirical findings reveal uniformly lower metric values for the models under consideration, suggesting enhanced forecasting accuracy. To further substantiate these findings and draw more definitive conclusions regarding forecasting precision, we utilize the Diebold-Mariano (DM) test. This test serves as an objective framework for comparing the predictive performances of two competing forecasting models by evaluating the statistical significance of the differences in their forecasting errors. For this purpose, we adopt MSE as the primary loss function to quantify prediction accuracy.

The outcomes of the DM test affirm the superiority of LSTM models in forecasting narrow money demand (M1), with the test results being statistically significant at a 1% level of significance when contrasted with almost all other models examined. Conversely, when forecasting the broader money demand (M3), the LASSO model is the most accurate, surpassing the forecasting performance of other models.

In conclusion, our study highlights the fundamental importance of maintaining a stable MDF for effective monetary policy and the potential of ML models, particularly LSTM and LASSO, in achieving precise forecasts of money demand. These ML models prove to provide an accurate prediction by unraveling the intricate patterns and dynamics essential for accurate money demand forecasting.

Appendix

Table 6.

Description of the variables

Variable	Variables description	Source
M1	Narrow monetary aggregate	CEIC
M3	Broad monetary aggregate	CEIC
CPI	Consumer Price Index	CEIC
IPI	Index of Industrial Production	CEIC
Govt securities yield	Government Securities Yield	CEIC
Call money rate	Call money rate	CEIC
NEER	Nominal effective exchange rate	CEIC
BSE mkt cap	Bombay Stock Exchange market capitalization	CEIC

Table 7.

Prediction of M1

	MSE	RMSE	MAPE	SMAPE	TIC
K = 2					
AR (1)	0.00163	0.04039	0.27357	0.27410	0.00164
RF	0.00407	0.06378	0.43521	0.43654	0.00260
GB	0.00561	0.07489	0.51580	0.51763	0.00305
XGBoost	0.00478	0.06916	0.50488	0.50644	0.00282
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	0.00067	0.02587	0.18430	0.18408	0.00105
K = 3					
AR (1)	0.00422	0.06495	0.44968	0.45106	0.00265
RF	0.00759	0.08711	0.61506	0.61755	0.00355
GB	0.00814	0.09024	0.63670	0.63934	0.00368
XGBoost	0.00210	0.04584	0.30681	0.30740	0.00187
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	0.01786	0.13365	0.96713	0.96133	0.00540
K = 4					
AR (1)	0.00490	0.07002	0.48359	0.48520	0.00285
RF	0.00781	0.08837	0.61852	0.62107	0.00360
GB	0.01006	0.10028	0.70007	0.70336	0.00409
XGBoost	0.00588	0.07666	0.53900	0.54091	0.00312
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	0.38360	0.61935	4.36226	4.24103	0.02463
K = 5					
AR (1)	0.00481	0.06932	0.48069	0.48226	0.00282
RF	0.00798	0.08933	0.62454	0.62716	0.00364
GB	0.00846	0.09200	0.65147	0.65425	0.00375
XGBoost	0.00492	0.07012	0.49278	0.49438	0.00286
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	1.16807	1.08077	7.58210	7.22144	0.04230
K = 7					
AR (1)	0.00439	0.06625	0.45983	0.46127	0.00270
RF	0.00838	0.09154	0.64145	0.64420	0.00373
GB	0.00675	0.08218	0.57166	0.57387	0.00335
XGBoost	0.00464	0.06813	0.48441	0.48592	0.00278
SVR	0.00285	0.05338	0.36694	0.36601	0.00217
LASSO	0.04045	0.20113	1.41856	1.43188	0.00824
LSTM	0.07542	0.27463	1.94317	1.91890	0.01106

Table 8.

Prediction of M3

	MSE	RMSE	MAPE	SMAPE	TIC
			K = 2		
AR (1)	0.00764	0.08740	0.51865	0.51666	0.00317
RF	0.00572	0.07562	0.44030	0.43880	0.00275
GB	0.00583	0.07635	0.46145	0.45993	0.00277
XGBoost	0.00801	0.08950	0.51842	0.51632	0.00325
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.00687	0.08286	0.49565	0.49385	0.00301
			K = 3		
AR (1)	0.00582	0.07631	0.45038	0.44886	0.00277
RF	0.00381	0.06174	0.34802	0.34702	0.00224
GB	0.00532	0.07294	0.42409	0.42270	0.00265
XGBoost	0.00791	0.08891	0.08891	0.56428	0.00323
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.00370	0.06085	0.35110	0.35016	0.00221
			K = 4		
AR (1)	0.00462	0.06794	0.39716	0.39596	0.00247
RF	0.00481	0.06933	0.39223	0.39097	0.00252
GB	0.00398	0.06308	0.34811	0.34708	0.00229
XGBoost	0.00742	0.08617	0.51045	0.50851	0.00313
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.01052	0.10256	0.67720	0.67444	0.00372
			K = 5		
AR (1)	0.00432	0.06574	0.38390	0.38277	0.00239
RF	0.00325	0.05701	0.31400	0.31315	0.00207
GB	0.00308	0.05549	0.30813	0.30733	0.00202
XGBoost	0.00594	0.07708	0.47436	0.47281	0.00280
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.00212	0.04599	0.27226	0.27176	0.00167
			K = 7		
AR (1)	0.00413	0.06423	0.37380	0.37272	0.00233
RF	0.00334	0.05778	0.32976	0.32889	0.00210
GB	0.00361	0.06010	0.33609	0.33515	0.00218
XGBoost	0.00737	0.08587	0.52506	0.52313	0.00312
SVR	0.00480	0.06929	0.40566	0.40440	0.00252
LASSO	0.01778	0.13333	0.86918	0.87388	0.00487
LSTM	0.00312	0.05587	0.31937	0.31857	0.00203

Table 9.

LSTM Hyperparameters Grid for M1

K	Hidden Size	Number of Layers	Dropout Rate	Learning Rate
2	64	2	0.2	0.0005
3	64	1	0.2	0.0005
4	128	1	0.1	0.0001
5	64	3	0.1	0.0005
6	128	1	0.1	0.0005
7	32	1	0.1	0.003

Table 10.

LSTM Hyperparameters Grid for M3

K	Hidden Size	Number of Layers	Dropout Rate	Learning Rate
2	64	3	0.1	0.0008
3	64	2	0.2	0.0005
4	64	2	0.1	0.003
5	32	3	0.2	0.003
6	32	3	0.1	0.0008
7	32	3	0.1	0.0008

* *
*

References

Adil M.H., Haider S., Hatekar N.R. (2018) Revisiting Money Demand Stability in India: Some Post-Reform Evidence (1996–2016). *The Indian Economic Journal*, 66, 3–4, pp. 326–346.

Adil M.H., Hatekar N., Sahoo P. (2020) The Impact of Financial Innovation on the Money Demand Function: An Empirical Verification in India. Margin. *The Journal of Applied Economic Research,* 14, 1, pp. 28–61.

Aggarwal S. (2016) Determinants of Money Demand for India in Presence of Structural Break: An Empirical Analysis. *Business and Economic Horizons*, 12, 4, pp. 173–177.

Akinlo A.E. (2006) The Stability of Money Demand in Nigeria: An Autoregressive Distributed Lag Approach. *Journal of Policy Modeling*, 28, 4, pp. 445–452.

Arrau P., De Gregorio J., Reinhart C.M., Wickham P. (1995) The Demand for Money in Developing Countries: Assessing the Role of Financial Innovation. *Journal of Development Economics*, 46, 2, pp. 317–340.

Ball L. (2012) Short-Run Money Demand. *Journal of Monetary Economics*, 59, 7, pp. 622–633.

Bahmani-Oskooee M., Shabsigh G. (1996) The Demand for Money in Japan: Evidence from Cointegration Analysis. *Japan and the World Economy*, 8, 1, pp. 1–10.

Bahmani-Oskooee M., Bahmani S., Kones A., Kutan A.M. (2015) Policy Uncertainty and the Demand for Money in the United Kingdom. *Applied Economics*, 47, 11, pp. 1151–1157.

Bajari P., Nekipelov D., Ryan S.P., Yang M. (2015) Machine Learning Methods for Demand Estimation. *The American Economic Review*, 105, 5, pp. 481–485.

Barnett W.A., Ghosh T., Adil M.H. (2022) Is Money Demand Really Unstable? Evidence from Divisia Monetary Aggregates. *Economic Analysis and Policy*, 74, pp. 606–622.

Bordo M.D., Jonung L. (2003) Demand for Money: An Analysis of the Long-Run Behavior of the Velocity of Circulation. *New Brunswick and London: Transaction Publishing*.

Dekle R., Pradhan M. (1999) Financial Liberalization and Money Demand in the ASEAN Countries. *International Journal of Finance and Economics*, 4, 3, pp. 205–215.

Goulet Coulombe P., Leroux M., Stevanovic D., Surprenant S. (2022) How Is Machine Learning Useful for Macroeconomic Forecasting? *Journal of Applied Econometrics*, 37, 5, pp. 920–964.

Ghosh T., Agarwal S. (2021) Do Machine Learning Models Hold the Key to Better Money Demand Forecasting? *Environmental, Social, and Governance Perspectives on Economic Development in Asia*. Emerald Publishing Limited, pp. 21–36.

Gogas P., Papadimitriou T., Sofianos E. (2019) Money Neutrality, Monetary Aggregates and Machine Learning. *Algorithms*, 12, 7, p. 137.

James G.A. (2005) Money Demand and Financial Liberalization in Indonesia. *Journal of Asian Economics,* 16, 5, pp. 817–829.

King M. (2001) No Money, No Inflation The Role of Money in the Economy. *Économie Internationale*, 4, 4, pp. 111–131.

Laidler D.E. (1982) *Monetarist Perspectives*. Harvard University Press.

Nelson E. (2003) The Future of Monetary Aggregates in Monetary Policy Analysis. *Journal of Monetary Economics,* 50, 5, pp. 1029–1059.

Nguyen H.T., Nguyen D.T., Nguyen N.T., Nguyen H.M., Nguyen V.H., Tan N.N., ... Linh N.T.X. (2022) Macroeconomic Forecasting Based on LSTM-Conditioned Normalizing Flows. *Prediction and Causality in Econometrics and Related Topics*, pp. 658–669.

Pham T.T.X., Le T.D., Nguyen T.N. (2022) Neural Network Models for Inflation Forecasting: A Revisit. *Prediction and Causality in Econometrics and Related Topics*, pp. 152–168.

Thornton D.L. (2014) Monetary Policy: Why Money Matters (and Interest Rates Don't). *Journal of Macroeconomics*, 40, pp. 202–213.

Uyen P.H., Uyen V.T.L., Hoa L.T., Trung T.Q. (2022) LASSO Regression and its Application in Forecasting Macro Economic Indicators: A Study on Vietnam's Exports. *Prediction and Causality in Econometrics and Related Topics*, pp. 575–585.