REPORT

Cryptocurrencies Price Prediction

Contents

1 Introduction

Cryptocurrency is a form of digital currency that regulates the generation of currency units and verifies the transfer of funds using encryption techniques. Notably, cryptocurrencies are not governed by a central authority and operate on a decentralized structure. Since the launch of Bitcoin in 2009, cryptocurrencies have revolutionized the way people transfer money. Cryptocurrency was first proposed in 1998 by a computer scientist, Wei Dai, who developed a cryptography-based system that could be used to ease payments between parties. This system, called ”b-money,” laid the groundwork for future cryptocurrencies.

The systematic structural specification of Bitcoin [1] was published in November 2008 by an unknown individual or group using the alias Satoshi Nakamoto. Bitcoin was the first cryptocurrency to be decentralized. Since the introduction of Bitcoin in 2009, cryptocurrencies have transformed how money is sent and received. Bitcoin is still the most popular and valuable cryptocurrency in the world, despite the creation of thousands of other cryptocurrencies and several price fluctuations since then. At the time of this writing, Bitcoin’s market capitalization exceeds 475 billion US dollars. In addition, the market capitalization of all active cryptocurrencies, including Bitcoin, reaches 1.17 trillion US dollars [2].

Due to the decentralized nature of the majority of cryptocurrencies, their prices are not influenced by interest rates, inflation rates, or monetary policies, but rather by the perception of users based on news, websites, and other non-fundamental elements [3]. The stock markets are influenced by a variety of factors that create uncertainty, including political and economic issues that have a local or global impact. Understanding the success keys, or factors that provide accurate predictions, is a difficult task. We can examine the market using any technique, including technical indicators, price fluctuations, and market technical analysis. There is thus a need for automated prediction tools to assist investors in deciding whether to invest in bitcoin or other cryptocurrencies. Modern stock market predictions typically include automation technologies, and we could apply the same approach and strategy to this realm of cryptocurrency.

2 Machine Learning Technology

Machine Learning is a powerful and effective choice for trading strategies [4]. Its ability to uncover hidden data relationships that may elude human observation makes it invaluable in predicting numeric outputs like price or volume and identifying categorical outputs such as trends. By providing the model with heuristic input data, traders can leverage a wide array of machine learning models to gain insights and make informed trading decisions.

Several machine learning models have proven successful in trading. Regression models, including linear regression [5] and support vector regression [6], offer accurate price movement estimation based on historical data. Classification models like decision trees [7] and random forests [8] excel at identifying market trends and making categorical predictions. Neural networks, such as deep learning models [9], are highly adept at capturing complex patterns in financial data.

Extensive research has demonstrated the efficacy of machine learning in trading, with studies showing superior performance compared to traditional strategies and higher returns [10] [11]. Furthermore, machine learning techniques have been employed to analyze alternative data sources like social media sentiment [12] and news articles [13] to gain a competitive edge in the market.

Machine learning provides traders with a diverse set of models and techniques that enhance trading strategies. As technology continues to advance and more data becomes available, the role of machine learning in the financial markets is expected to grow significantly.

3 CryptoPredictions

In order to provide community with a platform in which different models and cryptocurrencies are available, we have designed a library named CryptoPredictions. Previous cryptocurrencies price forecasting papers used different metrics and dataset settings, which caused ambiguities and interpretation problems. To reduce those differences, we created a CryptoPredictions (a library with 8 models, 30 indicators, and 10 metrics).

CryptoPredictions could be a significant help because:

1. At the outset of our work, we faced a serious challenge of dataset scarcity. Many papers and repos fetched the data through different websites, such as Yahoo Finance. However, we have overcome this obstacle by using platforms such as Bitmex, which offer a common structure for different currencies.

2. Before the advent of our library, users had to run different codes for different models, making it difficult to compare them fairly. Fortunately, CryptoPredictions has made it possible to conduct a unified and equitable evaluation of different models.

3. With Hydra, users can easily structure and understand arguments, making it easier to run codes on different settings and check results. By using Hydra, users have a better understanding of the arguments. Furthermore, it is far easier to run a code on different settings and check the result.

4. While some models may perform exceptionally well in terms of accuracy, they often require a well-defined strategy for successful trading. Our back-tester can help users determine the effectiveness of the used model in real-world scenarios.

5. We understand that evaluating models can be challenging, which is why we offer a variety of metrics to help users measure progress towards accomplishing their tasks. By analyzing multiple metrics, it is possible to identify areas for improvement and correct what is not working. We will explain the pros and cons of each metric later in this report.

6. At CryptoPredictions, we do not fetch indicators from different websites, because it leads to problems such as null rows and the lack of information on indicators for all cryptocurrencies. Instead, CryptoPredictions calculates them in a way that doesn't carry the mentioned problems and could be generalized to other datasets.

4 Models

In this section, the information about different models that are used in the library can be obtained.

4.1 Random Forest

Random Forest is a machine learning technique that employs decision trees to define a prediction model [14]. Several decision trees are utilized by the Random Forest technique to forecast Bitcoin prices [15]. Random forests have been used to forecast the direction of a stock price trend. In 2016, Khaidem et al. employed random forests to forecast trends in the values of Apple, Samsung Electronics, and General Electric shares traded on the NASDAQ.

Specifically, Random Forest is an ensemble technique consisting of decision trees and bagging [16]. It causes different data samples to be used to train each tree for the same problem. Different trees view distinct chunks of the data while employing bagging. No tree is exposed to all training data. So, when integrating their results, certain inaccuracies are compensated for others, resulting in a more generalizable prediction [17].

Overfitting is a major issue that can have devastating effects on results; however, if there are enough trees in the forest, the Random Forest algorithm will not overfit the model. Moreover, missing values can be handled by the random forest. As previously stated, the random forest is based on a different technique known as bagging, which entails running a bootstrap on the experiment's data sampling on the input variables and employing a fixed number of variables.

A random sample of N cases from the training set is taken with replacement. This sample constitutes the training set to build tree i. Given that the number of input variables is M, the number of variables selected for each node is m ($m < M$), which remains constant during block generation. The block is then associated with the node by splitting it and utilizing the optimal division of its m properties. The number of predictors evaluated in each division is roughly equal to the square root of the total number of predictors. The most effective method for determining the appropriate value is to analyze the out-of-bag MSE for various

values of m. In general, if the variables chosen at each node are highly connected, small values of m result in favorable outcomes. As a result, at each node, m observations are used for training and M-m observations are used for testing. Globally, given a training set D of size n, m new training sets $D_{i=1, \ldots, m}$ of size n' are created from m samples with replacement. Afterwards, each decision tree is trained using the data set D_i. In addition, it enters the nodes of each tree in order to make a forecast when it has a new observation, as shown shown in Figure 1.

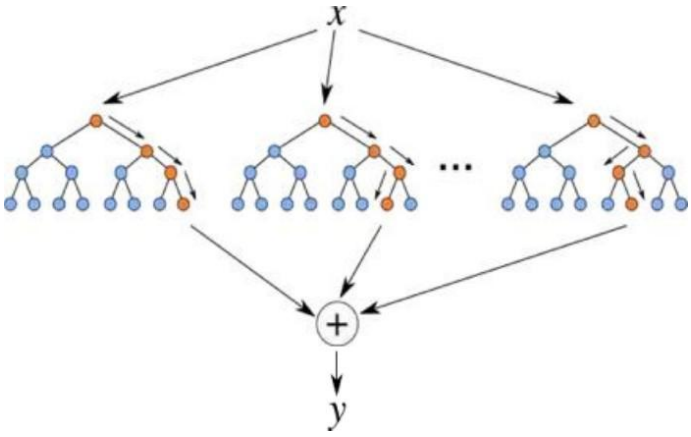

Figure 1: Structure of a random forest https://www.paradigmadigital.com/techbiz/machine-learning-dummies/

4.2 LSTM

LSTM (Long Short Term Memory) is another form of RNN module. Hochreiter and Schmidhuber (1997) [19] created LSTM, which was later developed and popularized by several researchers. The LSTM network consists of modules with recurrent consistency, similar to an RNN. The distinction between LSTM and RNN is the connectivity between the hidden layers of RNN. The RNN explanation structure is depicted in Figure. The only distinction between RNN and LSTM is the memory cell of the structure's hidden layer. And the design of three unique gates efficiently resolves gradient issues. Figure depicts the LSTM memory structure of the hidden layer [20].

Figure 2 explains that the RNN has deficiencies, which may be observed in the input. This problem was discovered by Bengio, et al. (1994) [21]. X_0, X_1 have a very large range of information X_t, X_{t+1}, so that when h_{t+1} requires information, those that are relevant to X_0, X_1 in the RNN are unable to learn to link information. Because the old memory that is saved becomes increasingly useless over time given that it is overwritten or replaced by new memory.

As it is shown in Figure 3, the LSTM's special units(recurrent hidden layers) contain memory blocks. In addition to memory cells with self-connections that store the network's temporal state, the memory blocks also contain multiplicative units called gates that regulate the flow of information. In the original architecture, each memory block comprised an input gate and an output gate. Controlling the flow of input activations into the memory cell is the input gate. The output gate regulates the flow of cell activations from the cell to the remainder of the network. Subsequently, the memory block received the forget gate [22].

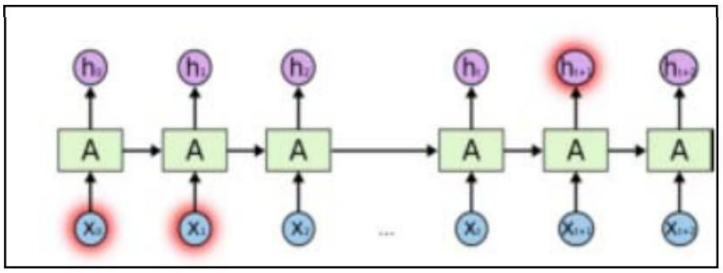

Figure 2: The Expanded Structure of RNN [18]

This addressed a shortcoming of LSTM models that prevented them from processing continuous input streams that were not divided into subsequences. The forget gate scales the internal state of the cell prior to adding it as input to the cell via its self-recurrent link, thereby forgetting or resetting the cell's memory in an adaptive manner. In addition, the contemporary LSTM architecture includes peephole connections from its internal cells to the gates in the same cell in order to learn precise output timing [23].

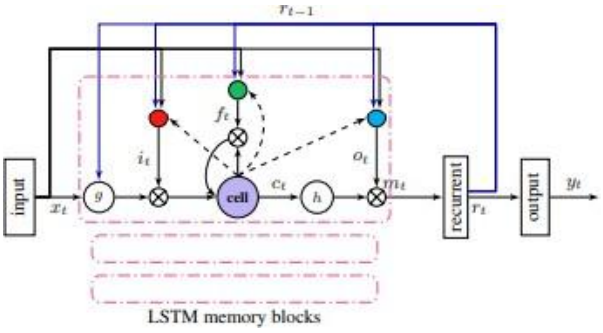

Figure 3: LSTM architecture: a single memory block is shown for clarity.

A mapping from an input sequence $x = (X_0, X_1, ..., X_t)$ to an output sequence $y = (Y_0, Y_1, ..., Y_t)$ is computed by an LSTM network through calculating the network unit activations iteratively from i=1 to t as follows:

In these formulas, σ, i, f, o and c are respectively the logistic sigmoid function, input gate, forget gate, output gate, and cell activation vectors, all of which are the same size as the cell output activation vector m. W denote as weight matrices (e.g. W_{ix} is the matrix of weights from the input gate to the input, W_{ic}, W_{fc}, W_{pc} are diagonal weight matrices for peephole connections), the b terms denote bias vectors (e.g. b_i is the input gate bias vector). Furthermore, g and h are the cell input and cell output activation functions, respectively, and tanh and are the network output activation functions, respectively, and softmax is present.

4.3 GRU

A gated recurrent unit (GRU) was presented [24] to enable each recurrent unit to capture adaptive dependencies on several time scales. Similar to the LSTM unit, the GRU possesses gating units that influence the flow of information inside the unit, but without distinct memory cells.

The architecture of the Gated Recurrent Unit:

In Figure 4, we have a GRU cell that is comparable to an LSTM cell or RNN cell.

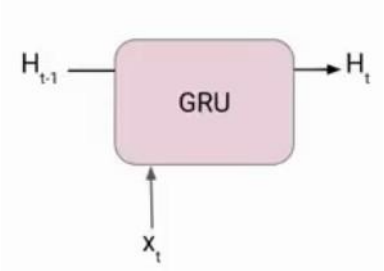

Figure 4: GRU architecture

At each timestamp t, it receives an input X_t and the preceding timestamp's hidden state H_{t-1}. Subsequently, it outputs a new hidden state H_t, which is consequently sent to the subsequent timestamp. Currently, a GRU cell consists mostly of two gates, as opposed to three gates in an LSTM cell. The initial gate is the Reset gate, while the second is the update gate [25].

Reset Gate (short-term memory)

The reset gate is responsible for the short-term memory of the network i.e the hidden state H_t. Here is the equation of the reset gate.

$$r_t = \sigma(x_t * U_r + H_{t-1} * W_r)$$

It resembles the LSTM gate equation. The sigmoid function limits r_t to (0,1) of weight matrices U_r and W_r.

Update Gate (long-term memory)

Similarly, we have an update gate for long-term memory and the equation of the gate is shown below.

$$u_t = \sigma(x_t * U_u + H_{t-1} * W_u)$$

The only difference is in the weight metrics, i.e., U_u and W_u.

Current Memory Gate

During a normal explanation of Gated Recurrent Unit Network, it is frequently disregarded. It is a component of the reset gate, just as the Input Modulation Gate is a subcomponent of the Input Gate, and is used to introduce nonlinearity to the input and make it zero-mean. A second reason to include it as a sub-part of the Reset Gate is to lessen the impact that past information has on the information that is being transmitted into the future. The calculation is as follows:

$$\hat{H}_t = tanh(x_t * U_g + (r_t \circ H_{t-1}) * W_g)$$

4.4 Orbit

Uber's Orbit is an open source package designed to ease time series inferences and forecasts using structural Bayesian time series models for real-world applications and scientific study [26].It employs probabilistic programming languages such as Stan [27] and Pyro [28] while providing a familiar and intuitive initialize-fit-predict interface for time series workloads.

It introduces a collection of refined Bayesian exponential smoothing models with a wide range of priors, model type specifications, and noise distribution options. The model includes a novel global trend term that is effective for short-term time series. Most significantly, it includes a well-crafted Python compute software/package named Orbit (Object-oriented Bayesian Time Series). The underlying MCMC sampling process and optimization are handled using the probabilistic programming languages Stan (Carpenter et al., 2017) and Pryo (Bingham et al., 2019). Pyro, created by Uber researchers, is a universal probabilistic programming language (PPL) built in Python and backed on the backend by PyTorch and JAX. Orbit presently has a subset of the available prediction and sampling algorithms for Pyro estimating.

4.5 Arima

The Autoregressive Integrated Moving Average (ARIMA) method was developed in 1970 by George Box and Gwilyn Jenkins and is also known as the BoxJenskins method [29]. The ARIMA method completely ignores independent variables while predicting, making it suited for interconnected statistical data (dependent) and requiring some assumptions such as autocorrelation, trend, or seasonality. The ARIMA method can predict historical data with the influence of difficult-to-understand data, has a high degree of accuracy in short-term forecasting, and can deal with seasonal data variations.

The ARIMA method is classified into four categories: Autoregressive (AR), Moving Average (MA), Autoregressive Moving Average (ARMA), and Autoregressive Integrated Moving Average (ARIMA) [30][31].

1. **Autoregressive (AR)**
   It was introduced by Yule in 1926 and expanded by Walker in 1932. This model assumes that data from prior periods is currently influencing current data. It's called autoregressive since it's rebuilt against the variable's prior values in this model. The AR method is used to calculate the order value of the coefficient p, which represents a value's dependence on its previous nearest value [32].

   The the general form of an AR model with order p (AR (p)) or an ARIMA model (p, 0, 0)is as follows:

$$X_t = \mu + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + e_t$$

2. **Moving Average (MA)**
   It was first presented by Slutzky in 1973. The MA approach is used to calculate the order coefficient q, which accounts for the varied movement of the prior residual value [33]. The following is the generic form of the MA model with order q (MA(q)) or ARIMA model (0, 0, q).

$$X_t = e_t - \phi_1 e_{t-1} - \phi_2 e_{t-2} - \ldots - \phi_q X_{t-q}$$

3. **Autoregressive Moving Average (ARMA)**
   This model combines the AR and MA models. Assume that the current period data is influenced by prior period data and the preceding period's forced

value. The following are common forms of the AR and MA or ARIMA processes' models (p, 0, q).

$$X_t = \mu + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + e_t - \phi_1 e_{t-1} - \phi_2 e_{t-2} - \ldots - \phi_q X_{t-q}$$

4. **Autoregressive Integrated Moving Average (ARIMA)**
The ARIMA Model assumes that the data used must be stationary, which means that the average variation of the data is constant. Nonstationary data must first be transformed into stationary data using a differencing procedure. The ARIMA technique is a statistical viewpoint method represented by three parameters, the first of which is the data AR process of the previous period, which is taken and maintained later in the Integrated process making the data easier to predict. The common version of the ARIMA model (p, d, q) is as follows:

$$X_t = \mu + X_{t-1} + X_{t-d} + \ldots + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + e_t - \phi_1 e_{t-1} - \phi_2 e_{t-2} - \ldots - \phi_q e_{t-q}$$

4.6 SARIMAX

ARMA is the combination of the AR and MA models. Adding an integration operator to an ARMA model produces an ARIMA model. A SARIMAX model incorporates exogenous variables assessed at time t that influence the value of input data at time t and integer multipliers of seasonality [34]. The parameters required to define the SARIMAX model are listed in Table.

Symbol	Remark	Symbol	Remark
p	Number of time lags to regress on.	θ_i	Parameters of Moving Average.
β	Constant (Measured as deviations from its mean).	$\Theta(L)^p$	An order p polynomial function of L.
θ_i	Parameters of Moving Average.	d	Order of differencing used.
L	Lag Operator	y_t	Prediction Value
$\phi(L)^q$	An order q polynomial function of L.	n	Number of exogenous variables.
Δ^d	Integration Operator.	β_n	Coefficients of exogenous variables.
x_t^i	Exogenous variables defined at each time step t. For in	Δ_s^D	Differencing operator.
q	Number of time lags of the error term to regress on	$\phi(L^s)^Q$	An order Q polynomial function with seasonality of L.
ϵ_t	Gaussian White noise at time t. (Zero mean)		

Figure 5: SARIMAX, List of symbols and model parameters.

The formula for the Autoregressive Model AR(p) is:

$$y_t = \Theta(L)^p * y_t + \epsilon_t$$

The Moving Average Model MA(d) is represented as follows:

$$y_t = \phi(L)^q * \epsilon_t + \epsilon_t$$

Autoregressive Moving Average Model ARMA(p,q) can be written as:

$$y_t = \Theta(L)^p * y_t + \phi(L)^q * \epsilon_t + \epsilon_t$$

and the Autoregressive Integrated Moving Average Model ARIMA(p,d,q) is stated as:

$$y_t^{[d]} = \Delta^d * y_t = y_t^{[d-1]} - y_{t-1}^{[d-1]}$$

$$\Delta^d * y_t = \Theta(L)^p * \Delta^d * y_t + \phi(L)^q \Delta^d * \epsilon_t + \Delta^d * \epsilon_t$$

Finally, Seasonal Autoregressive Integrated Moving Average Model with Explicative Variable SARIMAX((p, d, q) * (P, D, Q) is stated as follows:

$$\Theta(L)^p * \vartheta(L^s)^p * \Delta_s^D * \Delta^d * y_t = \phi(L)^q * \phi(L^s)^Q * \Delta^d * \Delta_s^D * \epsilon_t + \sum_{i=1}^{n} \beta_i * x_t^i$$

4.7 Prophet

Prophet is a method for forecasting time series data based on an additive model in which non-linear trends are fitted with annual, weekly, and daily seasonality, in addition to holiday effects. It is most effective when applied to time series with substantial seasonal effects and multiple seasons of historical data. Prophet is robust to missing data and fluctuations in the trend, and it typically handles outliers well [35].

$$p(y_t, y_{t-1}, ..., y_1) =$$
$$\aleph(y_t|m(t), \sigma^2).\aleph(y_{t-1}|m(t-1), \sigma^2)...\aleph(y_1|m(1), \sigma^2)$$

As long as the Prophet accurately displays the conditional mean and conditional variance, it should function adequately. Mathematically, we have this formula:

$$m_{prophet}(t+h) \approx \mathrm{E}[y_{t+h}|y_t, ..., y_1]$$
$$v_{prophet}(t+h) = \sigma^2 \approx \mathrm{V}ar[y_{t+h}|y_t, ..., y_1]$$
$$For\ all\ forecast\ periods\ t+h$$

This could be the case if the underlying system is in a condition of equilibrium, such as when the economy is stable. Therefore, once a significant shock occurs, the variance criterion will very certainly be violated. This is precisely what we observed in the preceding time series example.

4.8 XGBOOST

Gradient boosting (GBM) trees do unsupervised learning, in which they learn from data without a specified model. XGBoost is a popular gradient-boosting library. It can be used for GPU training, distributed computing, and parallelization. It is accurate, adaptable to all forms of data and situations, well-documented, and extremely user-friendly.

XGBoost is an abbreviation for Extreme Gradient Boosting. It is a properly parallelized and optimized version of the gradient boosting technique. Parallelizing the entire boosting procedure drastically reduces training time.

Rather than training the best possible model on the data (as is the case with conventional approaches), they trained hundreds of models on various subsets of the training dataset and then conducted a vote to determine the model with the best performance.

In many situations, XGBoost is superior to conventional gradient-boosting methods. The Python implementation provides access to a huge array of inner parameters that can be modified to improve precision and accuracy.

Parallelization, regularization, non-linearity, cross-validation, and scalability are some of XGBoost's most essential characteristics.

The XGBOOST algorithm works in such a way that it considers or estimates a function. To begin, we generate a sequence based on the function gradients. The following equation models a specific type of gradient descent. It specifies the direction in which the function decreases, as it represents the loss function to minimize. corresponds to the learning rate in gradient descent and is the rate of change fitted to the loss function. is anticipated to replicate the loss's behavior adequately.

$$F_{x_t+1} = F_{x_t} + \epsilon_{x_t} \frac{\partial F}{\partial x}(x_t)$$

To iterate over the model and determine its optimal formulation, we must describe the entire formula as a sequence and identify a function that will converge to the function's minimum. This function will serve as an error metric to help us minimize losses and sustain performance over time. The series approaches the minimal value of the function. This specific notation denotes the error function that applies while evaluating a gradient boosting regressor [36].

$$f(x, \vartheta) = \sum I(F((X_i, \vartheta), y_i))$$

5 Model Performance

In this section, the information about a common validation method for adjusting hyperparameters and different metrics for validating the prediction are provided.

5.1 Cross Validation

Cross-validation (CV) is a well-known method for adjusting hyperparameters and generating accurate measures of model performance. k-fold cross-validation and hold-out cross-validation are two of the most prevalent types of cross-validation [37].

First, we divided the dataset into the training set and the test set. If any parameters require tuning, we divide the training set into a training subset and a validation subset. On the training subset, the model is trained, and the parameters that minimize error on the validation set are selected. Finally, the model is trained on the complete training set using the selected parameters, and the error on the test set is recorded.

But there is a serious difference when we want to use cross validation in time series. Because of two reasons:

1. **Temporal Dependencies**
   Particular care must be used while separating time series data in order to prevent data leaking. To accurately imitate the "actual world forecasting environment, in which we stand in the present and predict the future"[27] , the forecaster must withhold any data regarding events that occur after the events used for fitting the model. Instead of k-fold cross-validation, we employ hold-out cross-validation for time series data, where a portion of the data (temporally separated) is reserved for assessing the model's performance. For example, observe the Split 4 in the above Figure 6 where the validation data which is Fold 4 comes before the Fold 5 which is the part of the training data.

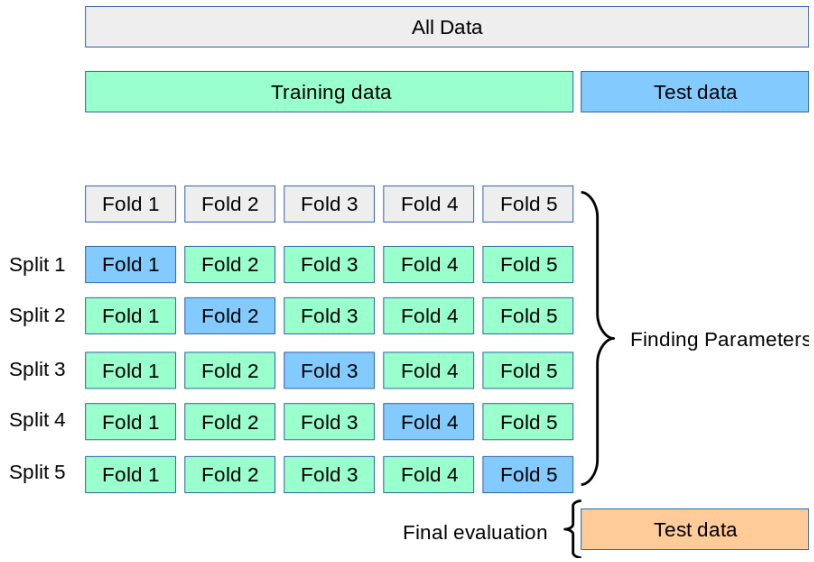

Figure 6

2. **Arbitrary Choice of Test Set**
   You may notice that the selection of the test set in Hold-out validation is somewhat random, which may imply that our test set error is an inadequate estimate of error on a separate test set.

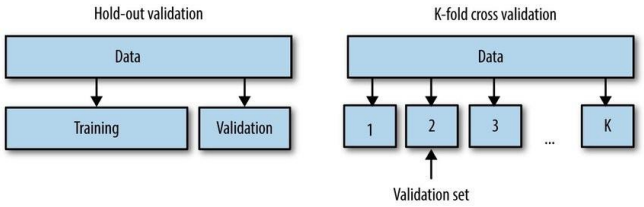

Figure 7

To overcome this, we employ the Nested Cross-Validation approach. Nested CV has an outside loop for error estimates and an inner loop for parameter adjustment (see the Figure below). The inner loop operates precisely as previously described: the training set is divided into a training subset and a validation set, the model is trained on the training subset, and the parameters that minimize error on the validation set are selected. Nevertheless, we now include an outside loop that divides the dataset into numerous training and test sets, and the error on each split is averaged to obtain a robust estimate of model error. This is beneficial because a nested cross-validation procedure provides an almost unbiased estimate of the true error [38].

Method: After setting the number of splits, the training dataset is splitted to equal subsets. For instance, there are 6 equal subsets in the above figure. In the k^{th} iteration of the outer loop,the k first subsets are considered as a training subset and $k + 1^{th}$ is considered as a validation subset.

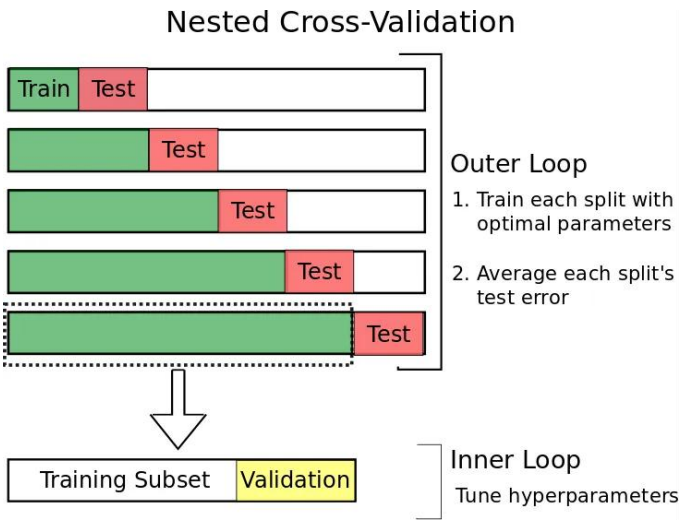

Figure 8

5.2 Metrics

After the obtaining the final predictions of the model, validating the data will usually be carried out by calculating [**?**]:

5.2.1 Mean Absolute Error (MAE)

It is the mean of the absolute value of the differences between the forecasting price and the actual value. It is easy to interpret and it benefits you by offering errors in the units of the data and the prediction. However, it does not penalize outliers(which could be not very important in price prediction). The most notable drawback of this metric is that it is scale dependent so we can not compare different cryptocurrencies with different units.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |A_i - F_i|$$

5.2.2 Mean Squared Error (MSE)

It is the mean of the square of the differences between the forecasting price and the actual value. In this metric, outliers are heavily punished. On the other hand, as the error is not in the original units of the data and prediction, it is harder to interpret. It is scale dependent as well so we have the similar issue like in MAE.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (A_i - F_i)^2$$

5.2.3 Root Mean Squared Error (RMSE)

It is the same as MSE apart from at the end we square root the result.In this metric, outliers are heavily punished like MSE and it has the strong point of being in the units of the data and prediction. It is kind of the best of both MSE and

MAE worlds. But, since you square the error, it could be still less interpretable. Furthermore, it is scale dependent.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(A_i - F_i)^2}$$

5.2.4 Mean Absolute Percentage Error (MAPE)

Mean absolute percentage error is the mean of the percentage difference between the actual value and prediction. This is often used as the baseline metric to measure most forecasting models. MAPE is not only easy to interpret but also scale independent which allows us to compare different cryptocurrencies. Nevertheless, in the cryptocurrencies with actual value near zero, we could have infinite error. In this metric, lower forecasts are bound to 100% error but higher forecasts could increase to infinite error, thus, it is biased to under-forecast.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left(100 \times \frac{|A_i - F_i|}{A_i}\right)$$

5.2.5 Symmetric Mean Absolute Percentage Error (SMAPE)

It is an extension of MAPE. It is the mean of the 200 times the difference between the actual value and prediction divided by the sum of their absolute values. It no longer favors the under forcastings. It is now fully bounded between 0% and 20%. Since the denominator could be still around zero, there is still a chance of infinite values. Additionally, interpreting a metric between 0% and 200% could be difficult. One supposed problem with SMAPE is that it is not symmetric since over and under forecasts are not treated equally. This is illustrated by the following example by applying the SMAPE formula:

- Over-forecasting: $A_t = 100$ and $F_t = 110$ give SMAPE = 9.09%
- Under-forecasting: $A_t = 100$ and $F_t = 90$ give SMAPE = 10.52%

$$SMAPE = \frac{1}{n}\sum_{i=1}^{n}\left(200 \times \frac{|A_i - F_i|}{|A_i| + |F_i|}\right)$$

5.2.6 Mean Absolute Scaled Error (MASE)

It has two modes. In case of no seasonality:

$$MASE = \frac{\frac{1}{n}\sum_{i=1}^{n} A_i - F_i}{\frac{1}{T-1}\sum_{t=2}^{T} A_t - A_{t-1}} = \frac{\sum MAE}{\frac{1}{T-1}\sum_{t=2}^{T} A_t - A_{t-1}}$$

In case of no seasonality:

$$MASE = \frac{\frac{1}{n}\sum_{i=1}^{n} A_i - F_i}{\frac{1}{T-m}\sum_{t=m+1}^{T} A_t - A_{t-m}} = \frac{MAE}{\frac{1}{T-m}\sum_{t=m+1}^{T} A_t - A_{t-m}}$$

This is the mean absolute scaled error for both seasonal and non-seasonal time series and is probably the best and most fair metric to use. This metric compares the output to the naive forecast.

Naive forecasts are the most cost-effective forecasting model, and provide a benchmark against which more sophisticated models can be compared. This forecasting method is only suitable for time series data. Using the naive approach, forecasts are produced that are equal to the last observed value. This method works quite well for economic and financial time series, which often have patterns that are difficult to reliably and accurately predict. If the time series is believed to have seasonality, the seasonal naive approach may be more appropriate where the forecasts are equal to the value from last season.

In time series notation: $\hat{y}_{T+h|T} = y_t$

In MASE, if the error is less than one, then it can be concluded that the forecast is better than an averaged naive forecast. On the contrary, if there is more than one, the forecast is worse than an averaged naive forecast. The advantages of this metric are scale independency and penalizing under and over forecasting equally.

5.2.7 Mean Squared Logarithmic Error (MSLE)

It measures the ratio or relative difference between the actuals and the forecasted values by calculating the mean of the square of the logarithm of the actual value plus one divided by the forecasted value plus one. This metric punishes the under forecasting more than over forecasting. Nonetheless, it is not easy to interpret.

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(\frac{A_i + 1}{F_i + 1}))^2$$

6 Result

To have a fair evaluation, we decided to compare different models on different cryptocurrencies and metrics. The training dataset is equal in all models which is from ’2022-11-13 13:30:00’ to ’2023-01-01 9:30:00’. Also, the test dataset is equal in all models as well and is from ’2023-01-01 10:30:00’ to ’2023-02-16 10:30:00’. We used hourly data and report the results in the below graphs.

6.1 Accuracy Score & F1-Score

As it can be seen in the Figure 9 and Figure 10, different cryptocurrencies’ graphs are close to each other. On roughly all the models, ARIMA and SARIMAX have the best results. In the second place, we have the Prophet, which has a superb result near ARIMA and SARIMAX. After that, we have Orbit whose result is not as good as other three models, however it is acceptable. Finally, we have XGBoost, Random Forest, LSTM, and GRU which have typical results close to each other around 55% in terms of accuracy and 0.5 in terms of F1-Score.

6.2 Recall Score & Precision Score

It is evident in Figure 11 & Figure 12, the results of SARIMAX, ARIMA, Prophet and Orbit indicate a considerable difference with others, and so we ob-

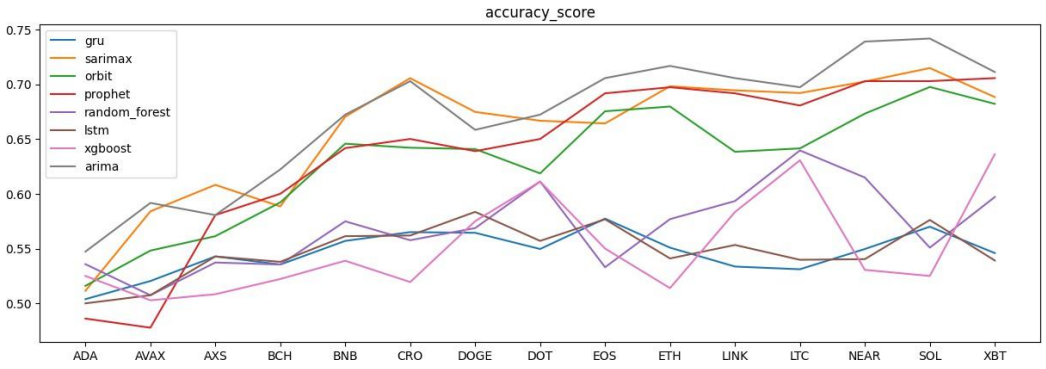

Figure 9

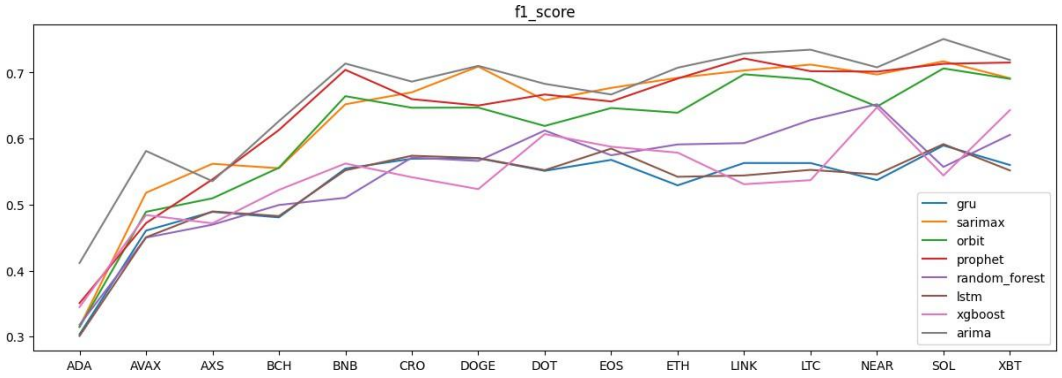

Figure 10

serve the same graph to the former. Meanwhile in these two metrics the results for ADA, AVAX and AXS show the lower score than others and it makes the task harder for prediction.

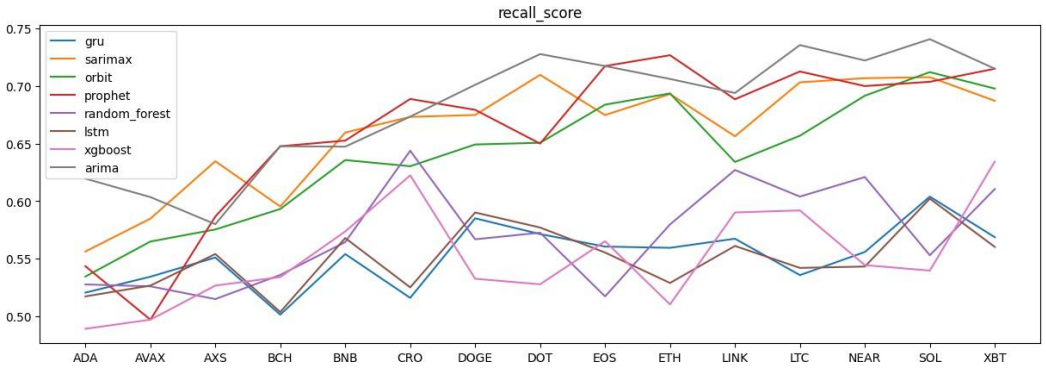

Figure 11

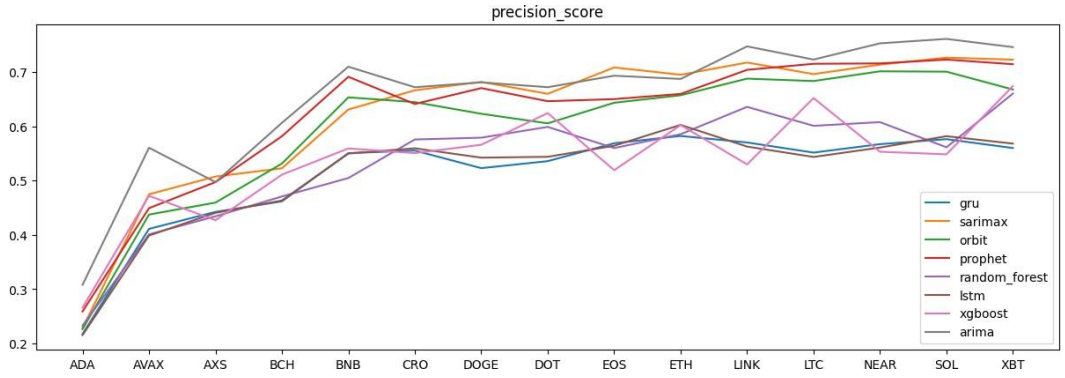

Figure 12

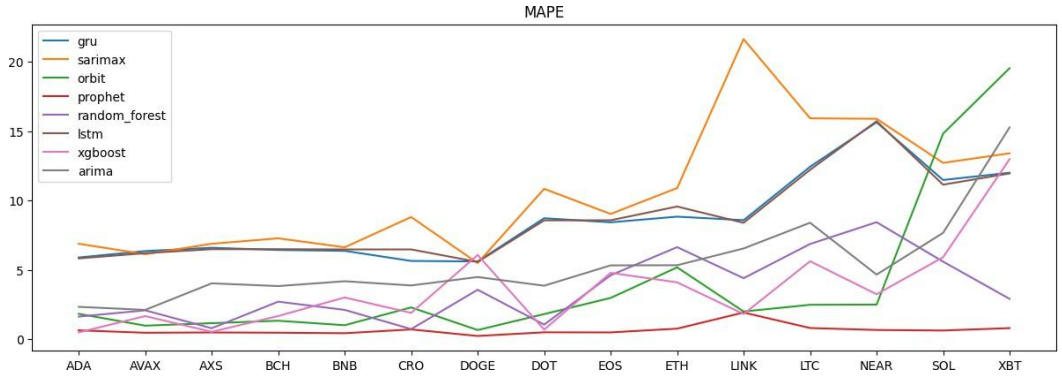

Figure 13

6.3 MAPE, SMAPE, MASE, and MSLE

Nevertheless, we have a different results with regard to MAPE, SMAPE, MASE, and MSLE. The results are shown in Figure 14 & Figure 15 & Figure 16. Despite the stunning result in terms of Accuracy and F1-Score, SARIMAX demonstrates a poor result. So it might not be a proper choice for price prediction. GRU and Random Forest has a poor result in the second place in these metrics. It can be seen that prophet shows the best result in all these metrics. On the other hand the results of Orbit, ARIMA, LSTM, and XGBOOST are placed between SARIMAX and Prophet.

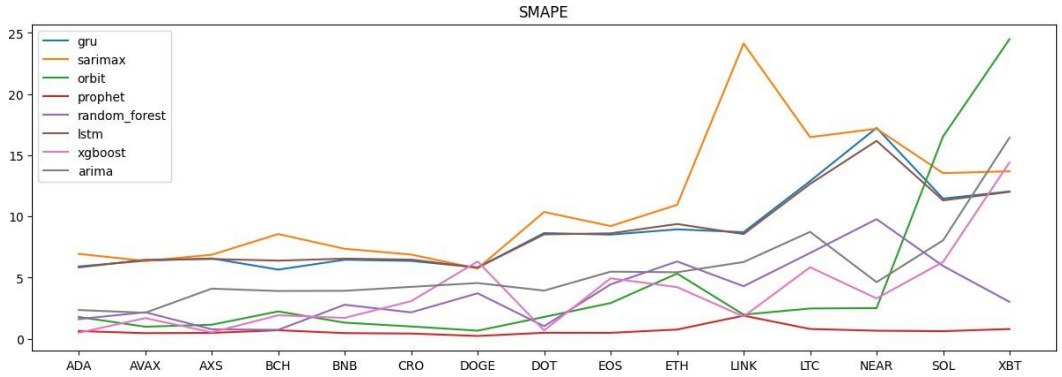

Figure 14

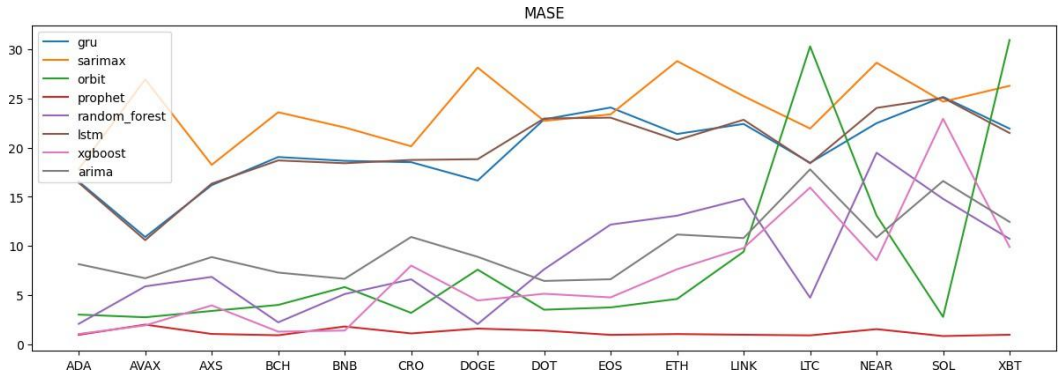

Figure 15

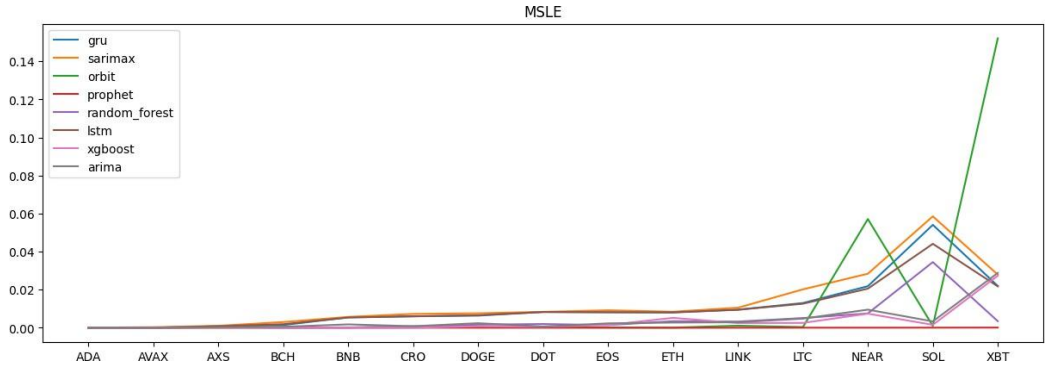

Figure 16

6.4 Results in Bitcoin

In this section, we conduct previous experiences particularly on BTC. The detail of the experiments is evident in Table 1.

Table 1: Experimental result for Second order low pass filter

	Accuracy	F1	Recall	Precision	MAE	RMSE	MAPE	SMAPE	MASE	MSLE
Prophet	0.71	**0.73**	**0.74**	0.73	**83**	**116**	**0.40**	**0.40**	**1.7**	**0.00004**
Orbit	0.68	0.70	0.69	0.70	137	184	0.65	0.66	2.7	0.00010
SARIMAX	0.70	0.70	0.69	0.71	1119	1391	5.48	5.74	24.6	0.00846
ARIMA	**0.72**	**0.73**	0.71	**0.75**	1009	1163	4.47	4.55	16.6	0.00277
XGBOOST	0.56	0.58	0.57	0.59	796	1115	3.60	3.76	15.1	0.00352
LSTM	0.54	0.54	0.53	0.56	1140	1434	5.56	5.81	25.1	0.00816
GRU	0.55	0.56	0.56	0.57	1140	1424	5.60	5.82	25.2	0.00804
Random Forest	0.58	0.59	0.58	0.61	787	1098	3.55	3.71	14.8	0.00347

In the BTC case, we can see similar results. The detail of the results is shown in Figure 14 to Figure 26.

6.5 Deduction

With respect to the previous graphs, it can be concluded that despite the superb result of SARIMAX and ARIMA, they have problems with predicting the price accurately. Meanwhile, Orbit works better in terms of MAPE, MAE,

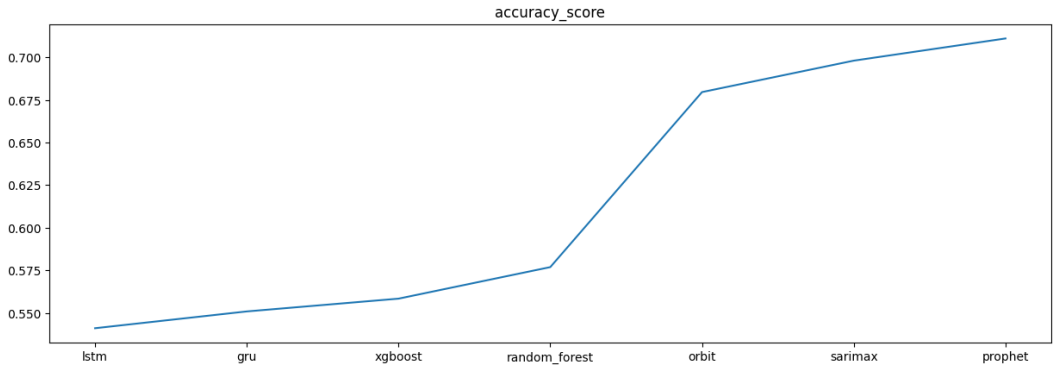

Figure 17

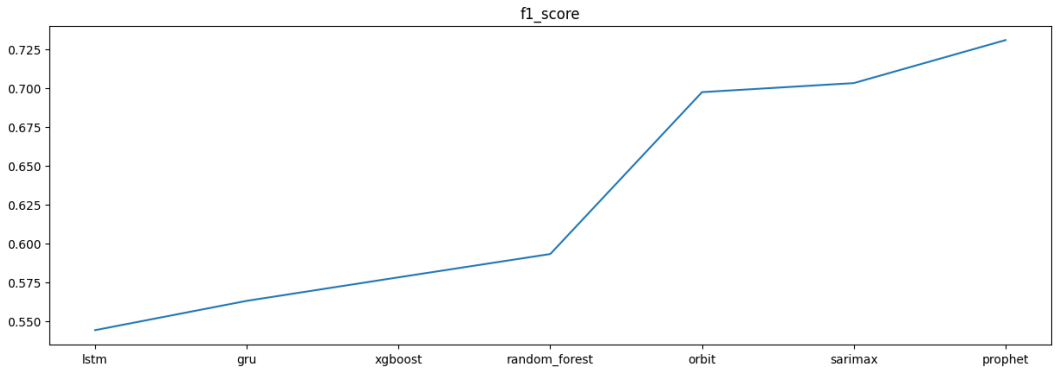

Figure 18

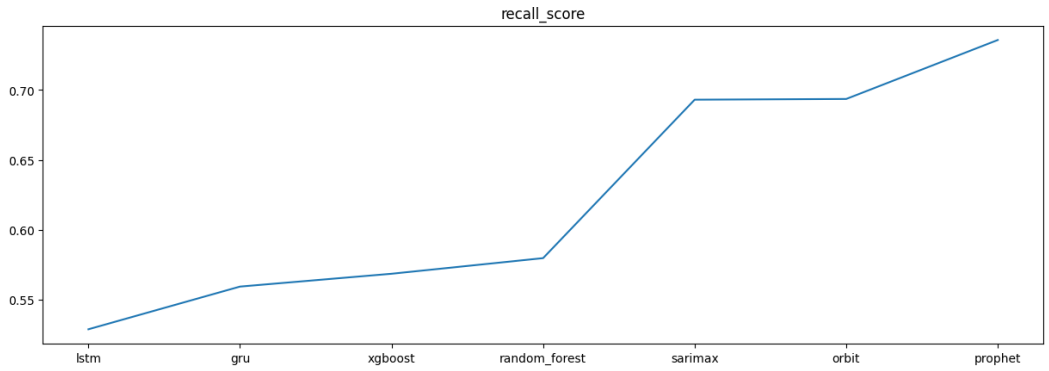

Figure 19

SMAPE, MASE, and MSLE. Lastly, Prophet not only shows a good performance in accuracy and F1-score but also demonstrates a stunning result in MAPE, MAE, SMAPEm and MSLE compared to other methods.

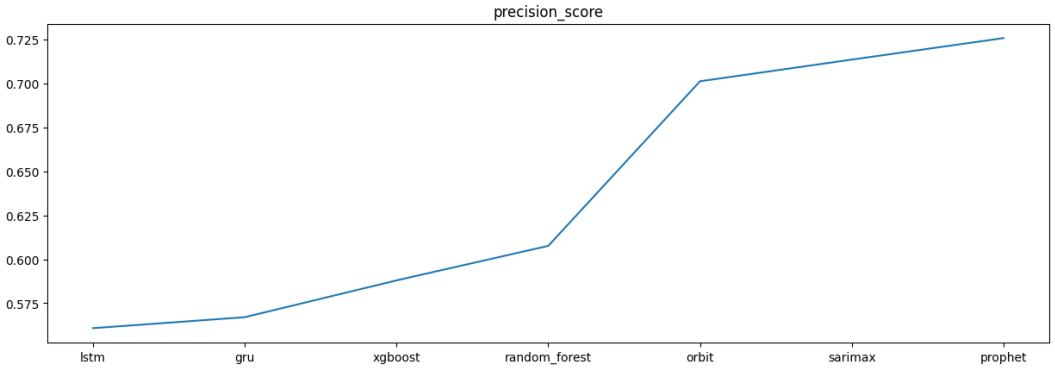

Figure 20

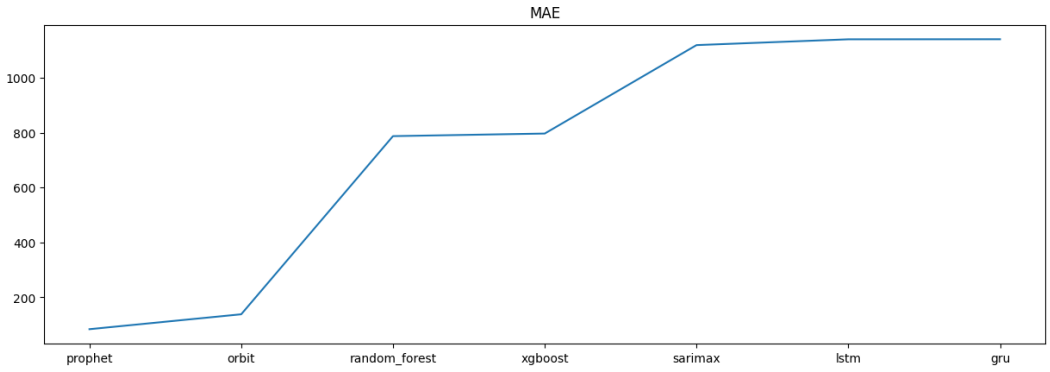

Figure 21

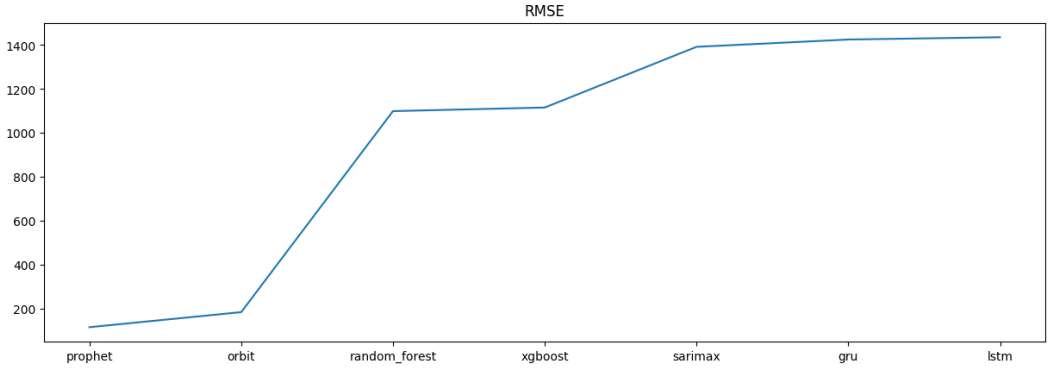

Figure 22

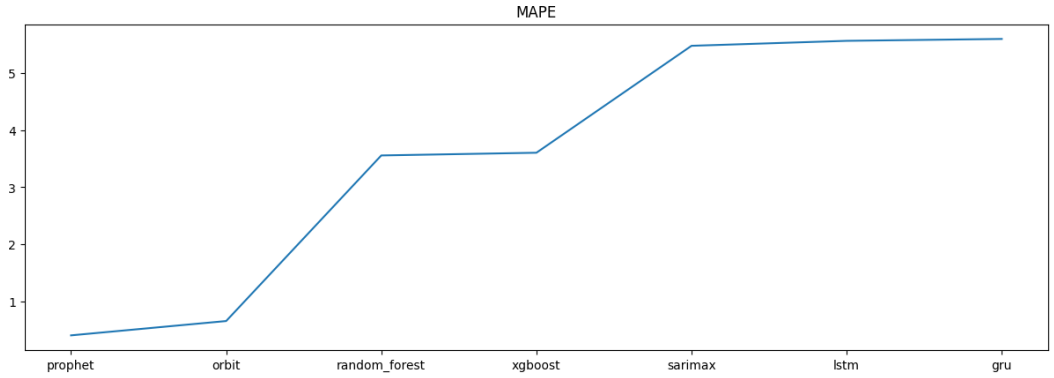

Figure 23

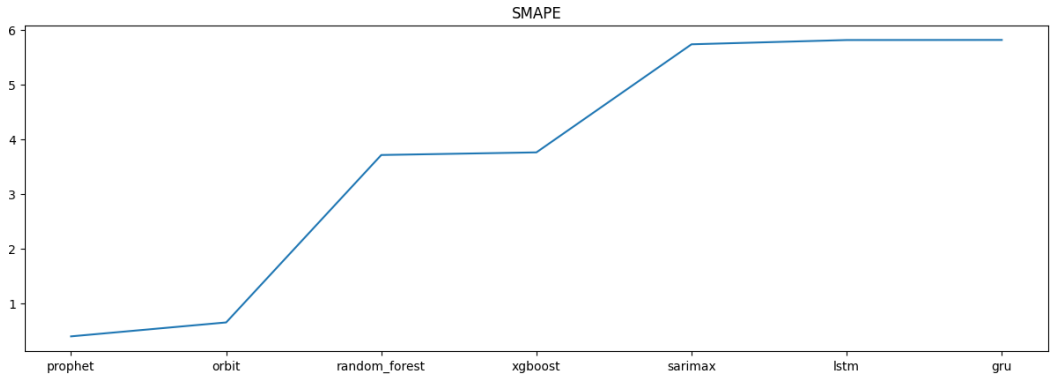

Figure 24

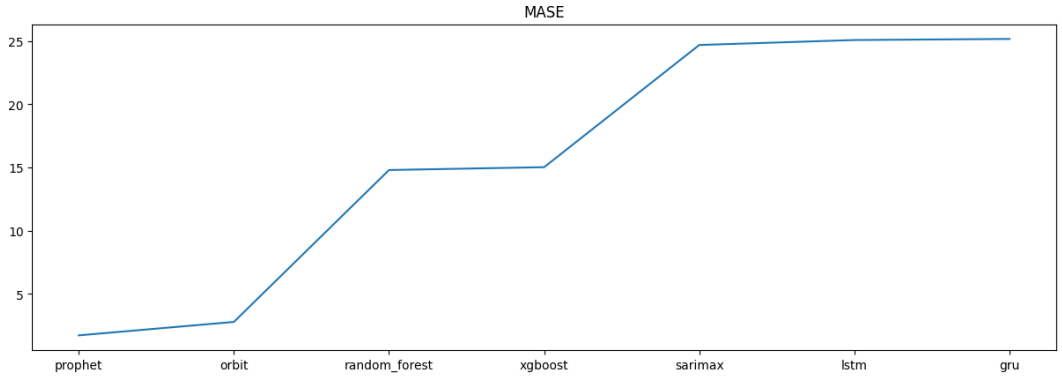

Figure 25

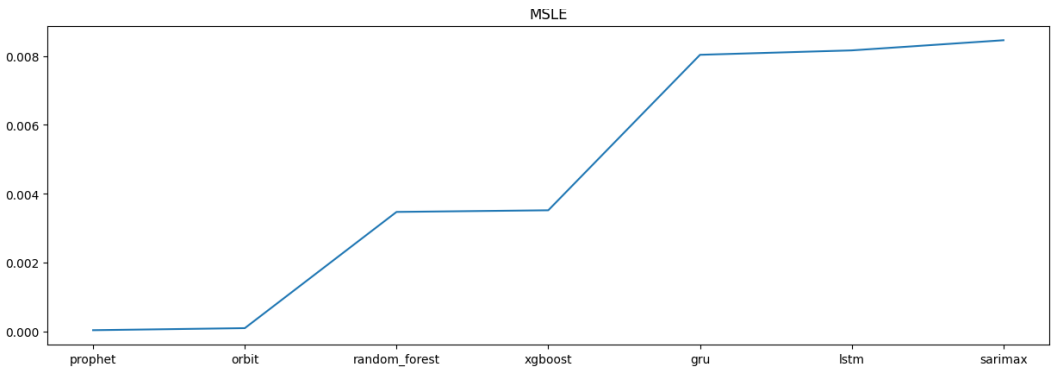
MSLE
0.008
0.006
0.004
0.002
0.000
prophet
orbit
random_forest
xgboost
gru
lstm
sarimax

Figure 26

7 Conclusion

In summary, this report explored various aspects of cryptocurrency forecasting, machine learning models, and evaluation metrics. The introduction provided an overview of cryptocurrencies, their decentralized nature, and their significant impact on the financial landscape.

The section on machine learning technology highlighted the suitability of machine learning models for cryptocurrency trading strategies, emphasizing their ability to uncover hidden data relationships.

The CryptoPredictions library was introduced as a valuable platform for cryptocurrency price forecasting, to overcome challenges such as dataset scarcity and the need for unified evaluation of different models. The library's features, including data collection, model evaluation, and indicator calculation, were outlined.

The models section covered several prominent models used for cryptocurrency forecasting, including Random Forest, LSTM, GRU, Orbit, ARIMA, SARIMAX, Prophet, and XGBOOST. Each model was briefly described, showcasing their unique characteristics and applications.

Lastly, the discussion delved into model performance evaluation, highlighting the importance of cross-validation for hyperparameter tuning and model selection. Various metrics, such as MAPE, MAE, SMAPE, MASE, MSLE, accuracy, and F1-score, were identified as essential tools for assessing the accuracy and effectiveness of the forecasting models.

Overall, while different models showed varying levels of performance in terms of accuracy and metrics, it was observed that Orbit and particularly Prophet consistently demonstrated strong results across multiple evaluation criteria. These models exhibited the potential to provide accurate and reliable cryptocurrency price predictions.

It is worth noting that the field of cryptocurrency forecasting is dynamic and evolving, and further research and experimentation are necessary to continually improve prediction accuracy and adapt to changing market conditions. The CryptoPredictions library and the models discussed in this exploration provide valuable tools and insights for researchers, traders, and investors seeking to navigate the world of cryptocurrency with greater confidence and understanding.

8 References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[2] "Coin Market Cap," [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/. [Accessed 09 02 2023].

[3] Makarov, I., & Schoar, A. (2020). Trading and arbitrage in cryptocurrency markets. Journal of Financial Economics, 135(2), 293–319.

[4] McNally, Sean, et al. "Predicting the Price of Bitcoin Using Machine Learning." 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2018, pp. 339–43.

[5] Geron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

[6] Drucker, H., Burges, C. J., Kaufman, L., Smola, A., Vapnik, V. (1997). Support vector regression machines. Advances in neural information processing systems, 9, 155-161.

[7] Breiman, L., Friedman, J., Stone, C. J., Olshen, R. A. (1984). Classification and regression trees. CRC press.

[8] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

[9] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[10] Zheng, H., Shi, J., Zhang, X., Li, F., Li, G. (2020). Deep reinforcement learning for stock trading: From models to reality. IEEE Transactions on Neural Networks and Learning Systems, 32(6), 2563-2575.

[11] Grootveld, M., Hallerbach, W. (2018). Machine learning for trading. The Journal of Portfolio Management, 44(3), 113-125.

[12] Bollen, J., Mao, H., Zeng, X. (2011). Twitter mood predicts the stock market. Journal of computational science, 2(1), 1-8.

[13] Ma, J., Gao, W., Fan, Y. (2020). News-driven stock market prediction using multi-scale deep neural networks. Expert Systems with Applications, 150, 113274.

[14] Mitchell, T. M. (1999). Machine learning and data mining. Communications of the ACM, 42(11), 30–36.

[15] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5–32.

[16] Breiman, L. (1996). Bagging predictors. Mach. Learn. 24, 123–140

[17] Biau, G. (2012). Analysis of a random forests model. The Journal of Machine Learning Research, 13(1), 1063–1095.

[18] M. W. P. Aldi, J. Jondri, and A. Aditsania, “Analisis Dan Implementasi Long Short Term Memory Neural Network Untuk Prediksi Harga Bitcoin,” eProceedings Eng., vol. 5, no. 2, 2018.

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735– 1780, 1997.

[20] F. Qian and X. Chen, “Stock Prediction Based on LSTM under Different Stability,” in 2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 2019, pp. 483–486.

[21] Y. Bengio, P. Simard, and P. Frasconi, “Learning Long-Term Dependencies with Gradient Descent is Difficult,” IEEE Trans. Neural Networks, 1994

[22] E. Kristensen, S. Østergaard, M. A. Krogh, and C. Enevoldsen, “Technical Indicators of Financial Performance in the Dairy Herd,” J. Dairy Sci., 2008.

[23] C. Scheier and W. Tschacher, “Appropriate algorithms for nonlinear time series analysis in psychology,” in Nonlinear dynamics in human behavior, World Scientific, 1996, pp. 27–43.

[24] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”

[25] https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/

[26] Edwin Ng, Zhishi Wang, Huigang Chen, Steve Yang, and Slawek Smyl. 2021. Orbit: Probabilistic Forecast with Exponential Smoothing. arXiv:2004.08492 [stat.CO]

[27] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee. “Stan : A Probabilistic Programming Language”

[28] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, Noah D. Goodman. “Pyro: Deep Universal Probabilistic Programming”

[29] E. B. Dagum, “The X-II-ARIMA seasonal adjustment method,” 2005.

[30] A. Hendranata, “ARIMA (Autoregressive Integrated Moving Average),” 2003.

[31] Y. S. Lee and L. I. Tong, “Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming,” Knowledge-Based Syst., vol. 24, no. 1, pp. 66–72, Feb. 2011.

[32] Hillmer, S. Craig, and George C. Tiao, “An ARIMA-Model-Based Approach to Seasonal Adjustment,” vol. 10, no. 1, pp. 5–24, 2017.

[33]] S. E. Said and D. A. Dickey, “Testing for unit roots in autoregressivemoving average models of unknown order.” pp. 599–607, 1984.

[34] From ar to sarimax: Mathematical definitions of time series models, https://phosgene89.github.io/sarima.html, journal=phosgene89.github.io.

[35] https://facebook.github.io/prophet/

[36] Tianqi Chen, Carlos Guestrin, “XGBoost: A Scalable Tree Boosting System”

[37] L. J. Tashman. Out-of-sample tests of forecasting accuracy: an analysis and review. International Journal of Forecasting, 16(4):437–450, 2000.

[38] S. Varma, R. Simon ”Bias in error estimation when using cross-validation for model selection”, 2006.