**Online learning with ELM**

**Abstract**

This article implements a machine learning model based on the Extreme Learning Machine (ELM) algorithm for predicting Bitcoin prices. Given the extreme volatility of Bitcoin prices, accurate predictions have become a significant challenge. This research includes stages of data collection, preprocessing, hyperparameter tuning, and model evaluation. The results indicate that the ELM model can effectively predict Bitcoin prices.

**Introduction**

Bitcoin, as the first and most well-known cryptocurrency, has attracted considerable attention. The extreme price volatility of this asset makes it an intriguing subject for analysis and prediction. Predicting Bitcoin prices can assist investors and traders in making better decisions. In this model, the ELM algorithm is utilized for predicting Bitcoin prices.

**What is Extreme Learning Machine (ELM)?**

Extreme Learning Machine (ELM) is a machine learning algorithm specifically designed for single-layer feedforward neural networks. Due to its high training speed and simplicity in implementation, it is recognized as an effective method for regression and classification problems.

**Online Learning**

Online learning refers to a machine learning approach where the model is continuously and gradually updated with new data. Instead of training on all data at once, this method allows the model to update itself as new data comes in, continuing its learning process.

**Data Collection**

Historical Bitcoin price data was collected from the CoinGecko API. This data includes open, high, low, and close prices over specified time intervals. For this project, data was collected from January 1, 2020, to the present day.

**Data Preprocessing**

After collecting the data, the following steps were performed for preprocessing:

* **Time Conversion**: Converting timestamps to a suitable format.
* **Price Change Calculation**: Using the closing price to create a target variable price\_change).
* **Missing Value Removal**: Eliminating any missing values from the dataset.

The choice of historical Bitcoin price data for the ELM model is due to the extreme price volatility of this cryptocurrency, easy access to data from reliable sources, and the ability to analyze market behavior. This data includes temporal information (such as open, high, low, and close prices) that helps identify price patterns and model the relationship between features and price changes. Additionally, using this data allows us to test our models on real data and evaluate their performance using various metrics.

**Implementation**

The ELM model is implemented as follows:

* **n\_hidden**: Number of hidden neurons.
* **W**: Weights between input and hidden layer.
* **b**: Biases of the hidden layer.
* **beta**: Weights between the hidden layer and output.

Here, **np.linalg.pinv(H)** computes the pseudo-inverse of matrix H. This inverse is then multiplied by the target values (y) to calculate the beta parameters. This step helps the model make the best predictions based on the outputs of the hidden layer.

Weights (W) and biases (b) are updated using the gradient descent method. The error is the difference between the target value (y\_i) and the model's prediction. **self.learning\_rate** determines how much the weights and biases should be updated. These formulas help the model gradually move towards the best predictions.

The error is calculated as the difference between the actual value (y\_i) and the model's prediction (np.dot(H\_i, self.beta)). This error helps the model understand how far it is from the actual prediction and update the weights and biases accordingly.

To convert inputs to outputs between 0 and 1, the Sigmoid activation function is used. The sigmoid function is a commonly used activation function in neural networks, and its main feature is that it can add non-linearity to the model, helping it learn more complex patterns. The sigmoid function is defined as:

[ \sigma(x) = \frac{1}{1 + e^{-x}} ]

The model is trained using the training data. At this stage, weights and biases are initialized randomly, and then the weights between the hidden layer and output are calculated using the matrix inverse method.

The model has the capability of online learning using the **partial\_fit** method. In this method, weights and biases are updated using gradient descent.

**Data Standardization**

The **StandardScaler** class from the scikit-learn library is used for standardizing the data. Standardization means that the features are transformed so that their mean is 0 and their standard deviation is 1. This process helps improve the performance of machine learning models, as many algorithms are sensitive to the scale of the data.

* **fit\_transform**: This method is used for the training data, calculating the necessary parameters (mean and standard deviation) and standardizing the data.
* **transform**: This method is used for the test data, standardizing it using the parameters calculated from the training data.

**Hyperparameter Tuning**

A dictionary named **param\_grid** is defined, which includes different values for the number of hidden neurons (n\_hidden). These values include 50, 100, 150, and 200. The goal of this step is to search for the optimal number of neurons to enhance the model's performance.

Next, an ELM model is created with an initial number of hidden neurons set to 100. This value is considered as a starting point, but it will be updated later using Grid Search.

**Grid Search Implementation**

In this section, Grid Search is performed to find the best number of hidden neurons:

1. **Initialization**: **best\_mae** is set to positive infinity to ensure that any calculated MAE will be less than this initial value. **best\_n\_hidden** is also initialized to None.
2. **Grid Search Loop**: For each value of **n\_hidden** in the **param\_grid** dictionary:

* A new ELM model is created.
* The model is trained using the training data.
* Predictions are made for the test data.
* The Mean Absolute Error (MAE) is calculated.
* If the calculated MAE is less than the best MAE found so far, the best MAE and the number of hidden neurons are updated.

After completing the Grid Search, the final model is trained with the best number of hidden neurons. This step allows us to utilize the optimal hyperparameters for training the model.

**Model Evaluation**

Next, predictions for the test data are made using the final model that has been trained with the best number of hidden neurons. These predictions help us evaluate the model's performance and see how close they are to the actual values.

**Results Analysis**

After training the model with the optimal number of hidden neurons, the prediction results are evaluated using the test data. Two main metrics are used to assess the model's performance: Mean Squared Error (MSE) and R² Score. These metrics help us gauge the accuracy of the model's predictions.

* **Mean Squared Error (MSE)**: This metric indicates the average of the squares of the errors in the model's predictions. The lower the MSE value, the better the model's performance.
* **R² Score**: This metric represents the proportion of variance in the dependent variable that is explained by the model. The R² value ranges between 0 and 1, and the closer it is to 1, the better the model's performance.

After executing the code and evaluating the model, the results indicate that the ELM model with the optimal number of hidden neurons has been able to provide accurate predictions of Bitcoin price changes. Notably, an R² Score close to 1 suggests that the model has explained a significant portion of the variance in the data.

The MSE is a performance metric that tells you how far the model's predictions are from the actual values. This value is calculated as follows:

[ MSE = \frac{1}{n} \sum\_{i=1}^{n} (y\_i - \hat{y}\_i)^2 ]

where (y\_i) are the actual values and (\hat{y}\_i) are the model's predictions.

In this research, a machine learning model based on the ELM algorithm was implemented to predict Bitcoin prices. By utilizing historical data and preprocessing techniques, the model achieved a satisfactory level of accuracy in its predictions. The results demonstrate that ELM can serve as an effective tool for predicting cryptocurrency prices.

**References**

* CoinGecko API Documentation
* Extreme Learning Machine: A New Learning Scheme for Feedforward Neural Networks
* Online Recurrent Extreme Learning Machine GitHub Repository
* GeeksforGeeks on Extreme Learning Machine
* Understanding LSTMs by Colah