HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Project II
Bitcoin Close Price Prediction

Instructor: **Prof.Le Tan Hung**

Student: **Nguyen Ngoc Khanh 20204915**

Hà Nội - 2023

Contents

Introduction

Cryptocurrencies have gained significant attention in recent years, with Bitcoin emerging as the leading and most well-known digital currency. As the popularity of Bitcoin continues to grow, there is an increasing interest in predicting its price movements. Accurately forecasting Bitcoin's close price can provide valuable insights for investors, traders, and researchers.

In this project, my objective is to build a simple web application powered by machine learning algorithms that allows user to predict the bitcoin close price

1. Dataset

CryptoCompare is a global cryptocurrency market data provider that was founded in 2014. It offers institutional and retail investors access to real-time, high-quality, reliable market and pricing data on 5,300+ coins and 240,000+ currency pairs.

The dataset made by using crytocompare python api contains 1001 bitcoin close prices from 2020-10-05 to 2023-07-02

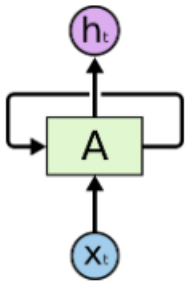
The dataset is divided into three sets training set, validation set and test set with the ratio 60%, 20%, 20% respectively in the chronological order

2. Related Works

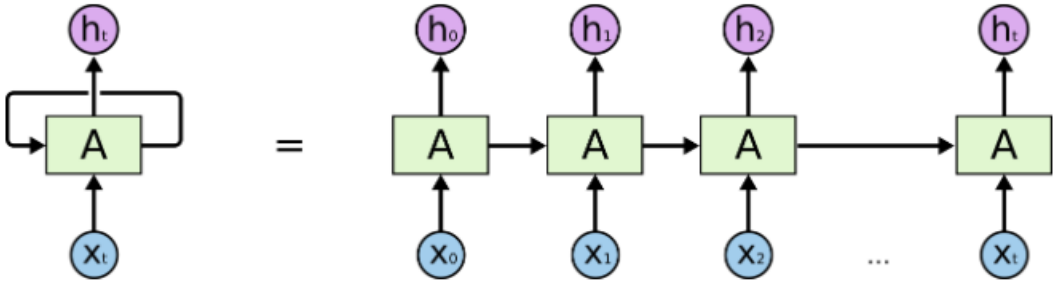
2.1. Long short term memory(LSTM)

2.1.1. Recurrent Neural Network (RNN)

RNN is a type of neural network that is capable of processing sequential input, including time-series data or text written in natural language. RNNs, in contrast to conventional feedforward neural networks, contain loops that let data from earlier inputs be stored and used to affect the current output.



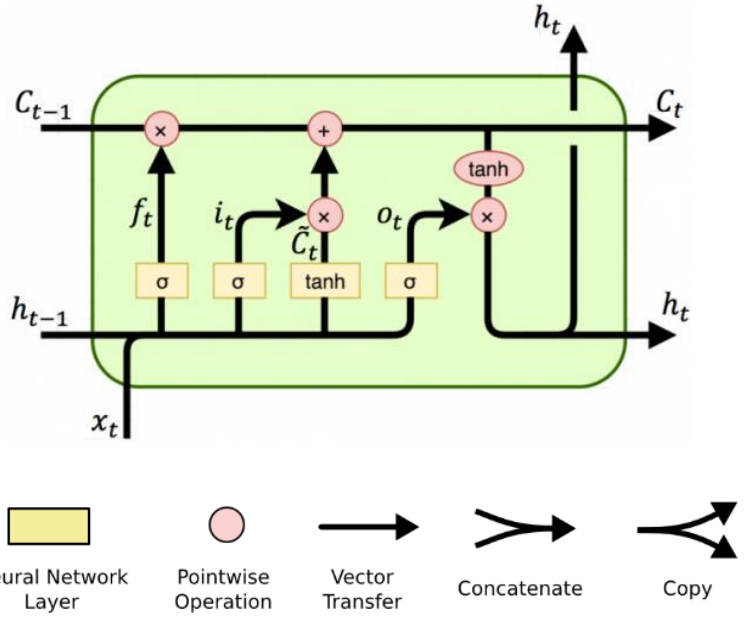
Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

2.1.2. LSTM cell

RNN in theory can handle long-term dependencies that are inherent in natural language text. However, in practice, that is not the case. Therefore, LSTM – a special type of RNN is developed to solve the problem.



Feed-Forward:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \tanh(C_t)$$

The core idea behind the success of LSTM over RNN is its cell state(C_t) which plays a role of the global or aggregate memory over all time-steps. While hidden state h_t cares more about the most recent time-step. Cell state corresponds to long-term memory and hidden state corresponds to short-term memory.

A LSTM cell consists of three gates:
+ Forget gate f_t decides which information to carry on
+ Input gate i_t decides which information to be updated in the cell state
+ Output gate decides which information to go out of a cell

2.2. Gradient Boosting

2.2.1. Tree ensemble

Given a dataset with n data points and m features $D = \{(x_i,\ y_i)\}$ $(|D| = n,\ x_i \in R^m,\ y_i \in R)$. A tree emsemble model comprises of K additive functions and is defined as:

$$\hat{y}_i = \phi(x_i) = \Sigma_{k=1}^{K} f_k(x_i), f_k \in F$$

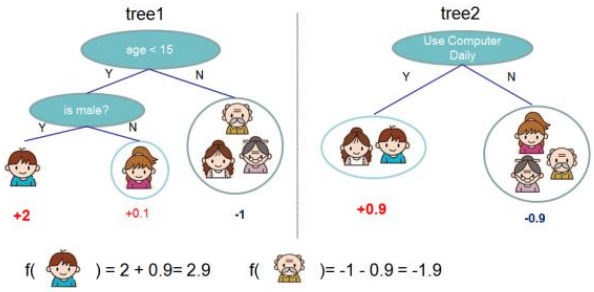
where $\hat{y}_i$ is the model prediction for a data point x_i

$F = \{f(x) = w_{q(x)}\}(q :\ R^m \to T, w \in R^T)$ is the space of regression tree

q is the structure of the tree that maps a data point to a leaf index

T is the number of leaves in the tree

w is the leaf weights



Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

2.2.2. Objective function

Our objective is to find a set of functions f_k , $k = \overline{1,K}$ that minimize the following cost function

$$L(\phi) = \Sigma_i l(y_i, \hat{y}_i) + \Sigma_k \Omega(f_k) \quad (1)$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$ that penalizes the complexity of the model

l is a differentiable convex loss function

2.2.3. Optimization

The tree ensemble model is optimized in an additive manner called gradient tree boosting. In the time step t, a new tree $f_t(x_i)$ is added to the model aiming to minimize the cost (1):

$$L^{(t)} = \Sigma_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

$$\approx \Sigma_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \Omega(f_t)$$

(second-order approximation)

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

Remove the constant operand $l\left(y_i, \hat{y}_i^{(t-1)}\right)$, minimize $L^{(t)}$is equivalent to minimize

$$\widetilde{L}^{(t)} = \Sigma_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right] + \gamma T + \frac{1}{2}\lambda \Sigma_{j=1}^{T} \mathrm{w}_\mathrm{j}^2$$

$$= \Sigma_{j=1}^{T} \left[\left(\Sigma_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\Sigma_{i\in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T$$

where $I_j = \{i | f_t(x_i) = w_j\}$

For a fixed tree structure $q(x)$, we can find the optimal weight $w_j^*$of leaf j by solving the equation

$$\frac{\partial \widetilde{L}^{(t)}}{\partial w_j} = 0$$

$$\Leftrightarrow \Sigma_{i\in I_j} g_i + \left(\Sigma_{i\in I_j} h_i + \lambda\right) w_j = 0$$

$$\Rightarrow w_j^* = \frac{-\Sigma_{i\in I_j} g_i}{\Sigma_{i\in I_j} h_i + \lambda}$$

Now, our work is turned into finding a tree structure that minimize

$$\widetilde{L}^{(t)} = -\frac{1}{2}\Sigma_{j=1}^{T} \frac{\left(\Sigma_{i\in I_j} g_i\right)^2}{\Sigma_{i\in I_j} h_i + \lambda} + \gamma T$$

We adopt a greedy algorithm to find the tree structure. We start by a single node and iteratively choose a leaf to split with the criterion that maximize the loss reduction

$$L_{split} = \frac{1}{2}\left[\frac{\left(\Sigma_{i\in I_L} g_i\right)^2}{\Sigma_{i\in I_L} h_i + \lambda} + \frac{\left(\Sigma_{i\in I_R} g_i\right)^2}{\Sigma_{i\in I_R} h_i + \lambda} - \frac{(\Sigma_{i\in I} g_i)^2}{\Sigma_{i\in I} h_i + \lambda}\right] - \gamma$$

where $I = I_L \, U \, I_R$

2.3.4. Skrinkage

To further prevent overfitting, we scale a newly added tree $f_t(x_i)$ by a factor η

$$\phi^t(x) = \phi^{t-1}(x) + \eta f_t(x)$$

Notice that η is very similar to the learning rate in the neural network

3. Experiment

All models are developed to minimize the mean absolute error(MAE) on the validation set

$$MAE = \frac{1}{n}\Sigma_{i=1}^{n}|y_i - \hat{y}_i|$$

3.1 Data preprocessing

Before being fetched to the model to train, data is scaled into the range of [0, 1] to gain the better performance of the optimization process

$$x_{scaled} = \frac{x - x_{max}}{x_{max} - x_{min}}$$

3.2. Long short term memory (LSTM)

The training process is done by Adam optimizer with learning rate = 1e-3

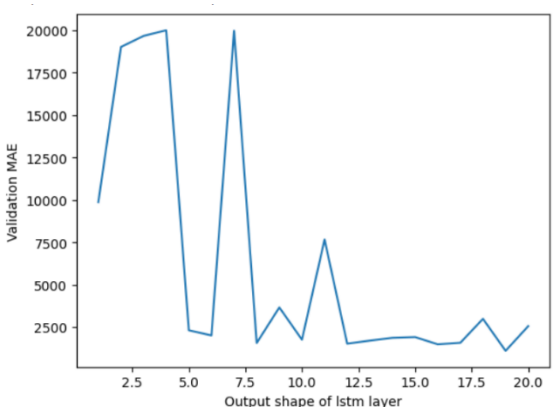
3.2.1 Architechture

LSTM: instead of using tanh activation, I use relu activation because the bitcoin close price is positive
Dropout: is used to regularize

Layer	Hyperparameters
LSTM	units = 19, activation = relu
Dropout	dropout_rate = 0.2
Fully connected layer	units = 1

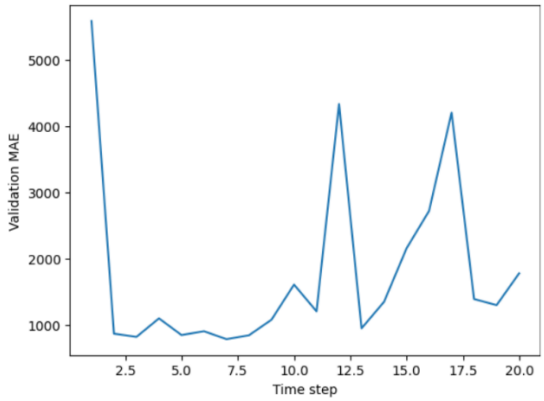
3.2.2 Hyperparameter tuning

Units: the output shape of the lstm layer
Value range: [1, 20]



We can see the downward trend of the line: the higher number of units tend to get better result
Minimum validation MAE is reached at 19 units

Time step: the number of previous and current bitcoin close prices to predict the next close price
Value range: [1, 20]



The line goes downward in some initial time steps, then go upward later
Minimum validation MAE is reached at 7 time steps

3.3. Gradient Boosting (XGBoost)

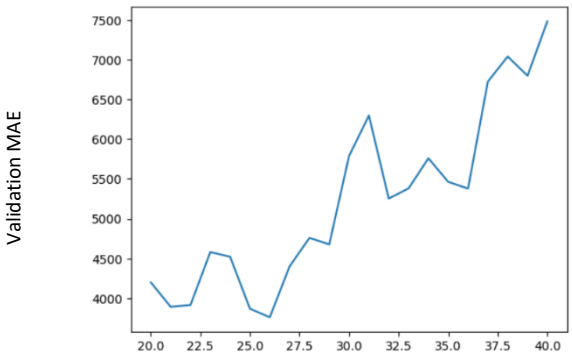
I make use of xgboost python package for gradient boosting algorithm

3.3.1. Hyperparameters

$l(y_i, \hat{y}_i) = \Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$ or squared error
n_estimators (number of trees) = 1000
time step = 26
All other hyperparameters are chosen to be the default values of XGBRegressor

3.3.2. Hyperparameters tuning

Time step: the number of previous and current bitcoin close prices to predict the next close price
Value range: [20, 40]



Time step

We can see the upward trend of the line: the higher number of time steps tend to get worse result
Minimum validation MAE is reached at 26 time steps

4. Final Result

All models are trained using tuned hyperparameters on the train and validation set, then evaluated on the test set

Model	MAE
LSTM	878.91
Gradient Boosting	821.32

The following two plots show the predictions of the next 100 days of two models

LSTM predictions

Gradient Boosting predictions

Conclusion:

- Gradient Boosting gives a better result than LSTM
- LSTM curve is smoother than Gradient Boosting curve. Therefore LSTM is more suitable to predict the trending of the price, while Gradient Boosting gives more accurate predictions

5. Web Application

5.1. Dependencies

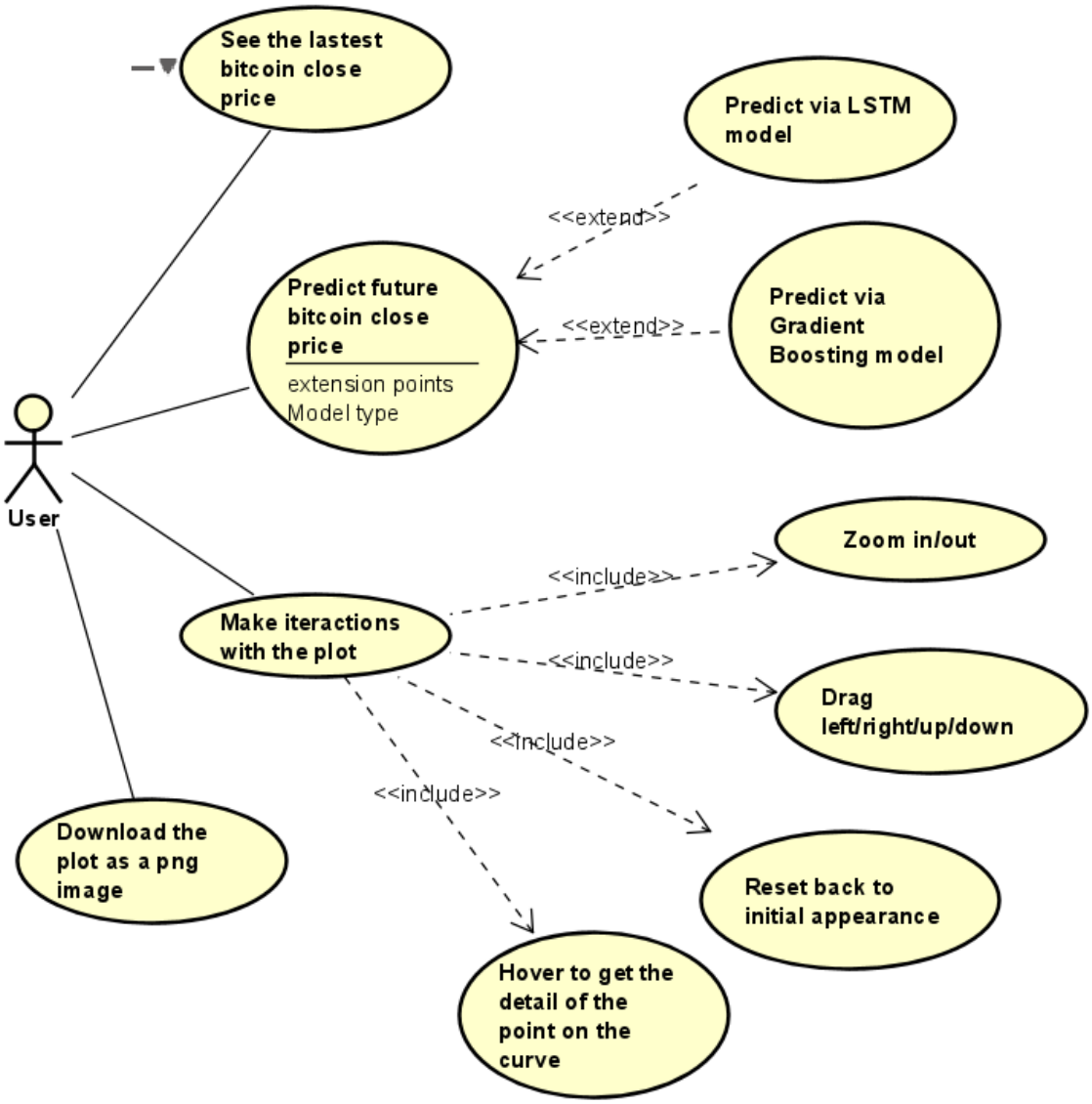
Frontend:

- HTML
- CSS
- Semantic UI

Backend:

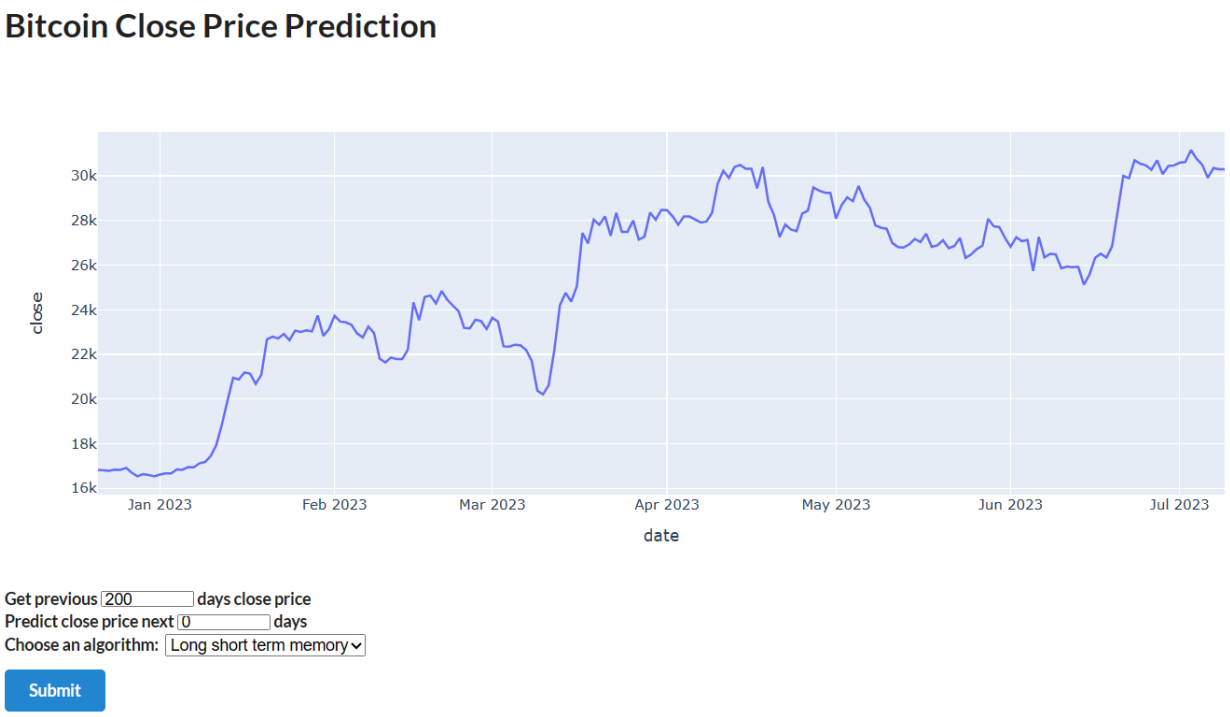
- Programming language: Python
- Server-side framework: Django
- Interactive visualization: Plotly
- Machine learning packages: xgboost, tensorflow, sklearn

5.2. UseCase diagram



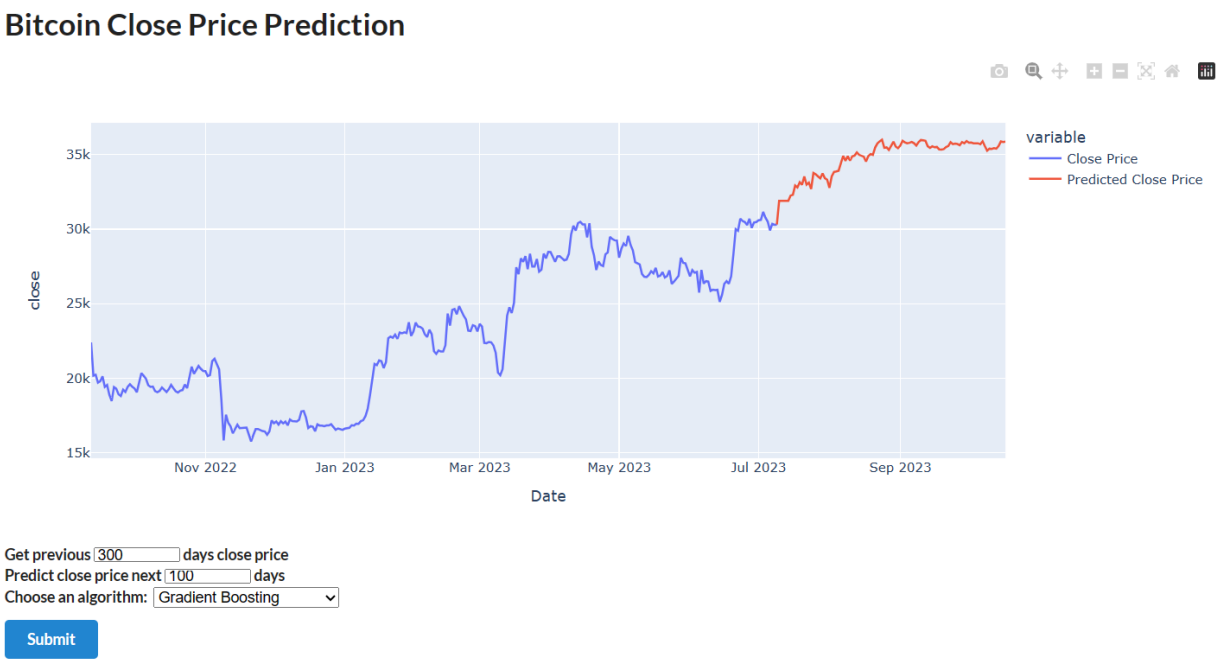
5.3. UI/UX design

Initial interface when user go into the web app



The plot shows the latest 200 previous bitcoin close prices and today bitcoin close price

When user choose gradient boosting to predict the next 100 bitcoin close prices, then click the submit button



References

[1] Long Short-Term Memory (Sepp Hochreiter and Jürgen Schmidhuber), In Neural Computation, volume 9, 1997
[2] Understanding LSTM Networks -- colah's blog
[3] XGBoost: A Scalable Tree Boosting System (Tianqi Chen, Carlos Guestrin) , 2016
[4] Django with Data Science | Data science Python course