FTEC 5530

Evaluating the Effectiveness of Machine Learning in Cryptocurrency Trading

Group 6 Members:

CHEN Angyu 1155197620
CHEN Yunfan 1155198241
HUANG Shiqi 1155204275
JIANG Tianyun 1155197611
LIANG Haojin 1155200065
NIU Xinyan 1155201239

Table of contents

01

Introduction

CHEN Yunfan

Motivation

- **Challenges with Traditional Models:**
  - Inability to capture temporal dependencies.
  - Difficulty handling non-linear relationships.
  - Limited adaptability to market changes.
  - Inadequate management of high-dimensional data.

VS

- **Advantages of ML Techniques:**
  - Superiority in capturing temporal dependencies with LSTM models.
  - Effective modeling of non-linear market dynamics.
  - Enhanced adaptability to evolving market conditions.
  - Efficient processing of high-dimensional data.

Literature Review

Features and Datasets for Cryptocurrency Prediction

Title	Model
Historical data	Open, high, low, and close prices along with volume are important factors for cryptocurrency prediction.
Intraday Price Prediction	A study suggests that using five lagged prices as inputs can effectively predict the next 5-minute price of Bitcoin. Including additional inputs may not be advantageous.
Additional Factors	Cryptocurrency data such as block size, hash rate, mempool transaction count, mempool size, Google Trend Search Volume Index, and gold spot price have been combined with historical data for price prediction.
Macroeconomic Indicators	Crude oil futures prices, gold futures prices, S&P500 futures, NASDAQ futures, and DAX index historical price data have been used along with Bitcoin historical prices to predict the direction, maximum, and minimum closing prices.
Technical Indicators	Crude oil futures prices, gold futures prices, S&P500 futures, NASDAQ futures, and DAX index historical price data have been used along with Bitcoin historical prices to predict the direction, maximum, and minimum closing prices.

Literature Review

Models for Cryptocurrency Prediction

Cryptocurrency	Recommended Models	Accuracy/Performance
Bitcoin	Linear regression	Outperform machine learning models for daily prices
	Machine learning models	Outperform statistical models for 5-min predictions
	SVM w/ radial basis kernel	Provides significantly better results
	LSTM	Highly recommendable for real-time price prediction
Ethereum	SVM w/ feature extraction	99%
	LSTM	67.20% for price prediction
	LSTM w/ technical indicators	99% for daily direction prediction
	LSTM w/ technical indicators + social network info	87% (with trading indicators)

02

Data Processing

JIANG Tianyun

Integrating Historical Prices and NVT Ratio

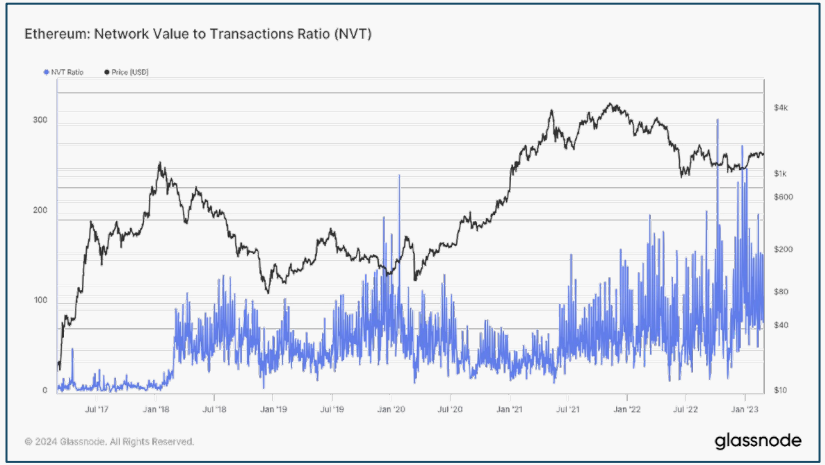
- **Data Source:** Glassnode Studio - A trusted provider of comprehensive Ethereum data.
- **Data Sets:**
  - **Training Set:** March 1, 2017, to July 3, 2019
  - **Testing & Backtesting Set:** July 3, 2019, to February 29, 2024
- **Historical Ethereum Pricing Data:**
  - **Details:** Daily Open, High, Low, Close (OHLC) prices in USD.
  - **Importance:** Foundation for predictive modeling and market performance insights.

```
df = pd.read_csv('../data/eth_price.csv')
df.drop(df.columns[1], axis=1, inplace=True)
df.columns = ['date', 'close', 'high', 'low', 'open']
df['date'] = pd.to_datetime(df['date'])
df = df[(df['date'] >= '2017-03-01') & (df['date'] <= '2024-02-29')]
df
```

- **Ethereum NVT Ratio:**
  - **Definition:** Network Value to Transactions Ratio.
  - **Function:** Evaluates Ethereum's valuation against transaction value, akin to the PE Ratio in equities.

```
df_nvt = pd.read_csv('../data/ethereum-network-value-to-transactions-ratio-nvt.csv')
df_nvt.columns = ['date', 'nvt ratio']
df_nvt['date'] = pd.to_datetime(df['date'])
df_nvt = df_nvt[(df_nvt['date'] >= '2017-03-01') & (df_nvt['date'] <= '2024-02-29')]
df_nvt
```

- **Introduction to NVT:**
  - Market Indication: Assesses potential market overvaluation or undervaluation.
- **Role of NVT Ratio:**
  - High NVT: Suggests possible speculative bubble； Low NVT: Indicates potential undervalued network status.
- **Data Merging Strategy:**
  - Objective: To create a unified dataset reflecting Ethereum's market dynamics.
  - Approach: Combines price data with NVT Ratio for a holistic analytical perspective.




```
df['NVT'] = df_nvt['nvt ratio']
df['NVT'] = df['NVT'].fillna(method='ffill')
df.to_csv('../data/input.csv')
df.set_index(['date'], inplace = True, drop=True)
list_feature = list(df.columns)
```

date	close	high	low	open	NVT
2017-03-01	17.758000	17.758000	16.970000	16.970000	7.576159
2017-03-02	19.208000	19.246000	19.144000	19.144000	5.164297
2017-03-03	19.610000	20.040000	19.610000	20.040000	5.427846
2017-03-04	18.702000	18.900000	18.660000	18.834000	8.371288
2017-03-05	19.200000	19.200000	19.150000	19.152000	9.768700
...	...	...	...	...	...

03

Model Comparison

CHEN Angyu

Comparism

Linear Regression Model

A statistical approach for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.

Pros: Simple, easy to understand and implement

Cons: May not capture non-linear characteristics and long-term dependencies of the data

Long Short-Term Memory Model

A type of Recurrent Neural Network (RNN), well-suited for time series prediction due to its ability to maintain a memory of past information and integrate it with new data inputs.

Pros: Able to learn long-term dependencies for complex time series data.

Cons: Complex, requiring more computational resources and data, longer training time

Linear Regression Model

```
In [17]: #plitting the  data into train and test set: 90% / 10%
         from sklearn import linear_model
         #dates = pd.DataFrame(np.arange(len(dataset)))
         #open = dataset['open']
         x_index = ['open','NVT']
         y_index = ['close']
         x_train, x_test, y_train, y_test = train_test_split(dataset[x_index], (

In [18]: model = linear_model.LinearRegression()
         # Fit linear model using the train data set
         model.fit(x_train, y_train)

Out[18]: LinearRegression
         LinearRegression()

In [19]: # The coefficient
         print('Slope: ', np.squeeze(model.coef_)[0])
         # The Intercept
         print('Intercept: ', model.intercept_)

         Slope:  1.0001152571367502
         Intercept:  [0.73508393]
```

- **Identify variables:** High price, low price, open price, and NVT.
- **Multicollinearity test:**
  A significant multicollinearity issue among the high price, low price, and open price.
- **Model construction:**
  Use open price and NVT;
  Fit the model on the training dataset;
  Estimate the parameters (slope and intercept) to minimize the error between predictions and the actual values.
- **Prediction:** Output the results of the test set.
- **Model Evaluation:** Calculate MSE, RMSE, R2 and Explained Variance.

Long Short-Term Memory Model

model = Sequential() model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1))) model.add(Dropout(0.2)) model.add(Dense(units=1))

```
In [15]: l_model = Sequential()


         l_model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.
         l_model.add(Dropout(0.2))

         l_model.add(LSTM(units=50, return_sequences=True))
         l_model.add(Dropout(0.2))

         l_model.add(LSTM(units=50, return_sequences=True))
         l_model.add(Dropout(0.2))

         l_model.add(LSTM(units=50))
         l_model.add(Dropout(0.2))


         l_model.add(Dense(units=1))

         l_model.compile(loss='mean_squared_error', optimizer='adam')
         history=l_model.fit(x_train, y_train, epochs=10, batch_size=32,validat
```

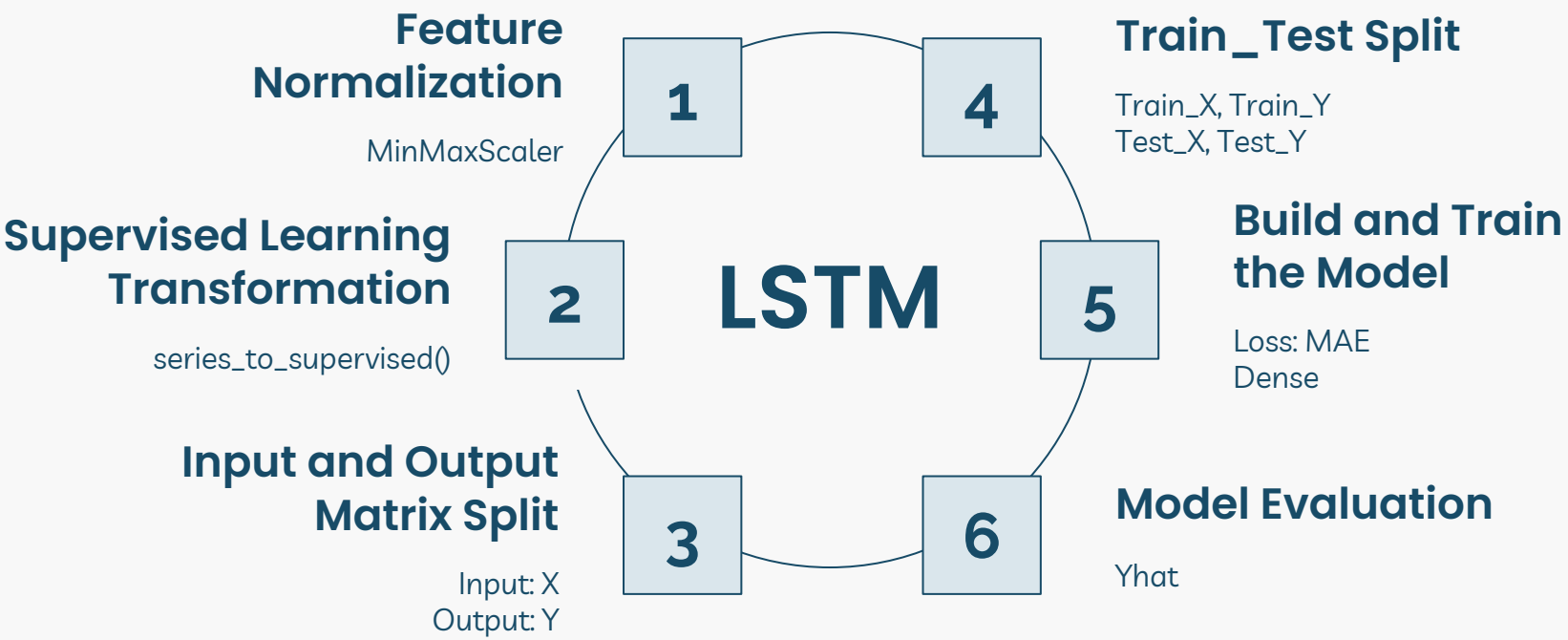
- **Data processing:**
  Divide the data into training and testing sets;
  Create MinMaxScaler instances to scale the features to the range of [0,1].
- **Model construction:**
  Create a serialized original time series data;
  Add the first LSTM layer to receive input data;
  Add a dropout layer to randomly discard 20% of neurons to reduce overfitting;
  Repeat the above two steps
- **Prediction:** Add dense layer, the output layer of the network model.
- **Model Evaluation:** Calculate MSE, RMSE, R2 and Explained Variance.

04

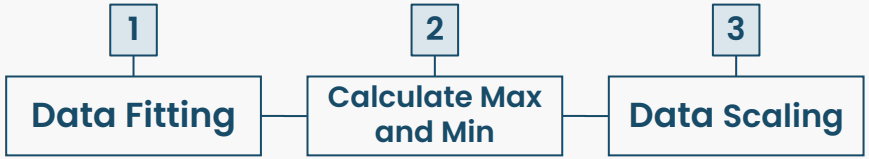
Model Training

NIU Xinyan

Multi-Factor LSTM Prediction Model



Step 1: Feature Normalization




```
values = df
values = values.astype('float32')

# Create a MinMaxScaler object to scale the data to (0, 1)
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit and transform operations on values using MinMaxScaler objects
scaled = scaler.fit_transform(values)
```

date	close	high	low	open	NVT
2017-03-01	17.758000	17.758000	16.970000	16.970000	7.576159
2017-03-02	19.208000	19.246000	19.144000	19.144000	5.164297
2017-03-03	19.610000	20.040000	19.610000	20.040000	5.427846
2017-03-04	18.702000	18.900000	18.660000	18.834000	8.371288
2017-03-05	19.200000	19.200000	19.150000	19.152000	9.768700
...	...	...	...	...	...
2024-02-25	3111.157214	3113.612582	3099.943825	3104.313420	107.155343
2024-02-26	3179.377544	3182.628043	3171.625989	3182.459859	66.200709
2024-02-27	3243.107970	3250.021864	3240.807238	3250.021864	58.230598
2024-02-28	3381.199751	3382.843713	3343.736769	3343.736769	54.721723
2024-02-29	3465.126544	3482.089781	3462.572413	3463.885691	67.301264

2557 rows × 5 columns

```
array([[2.4529244e-04, 1.4494103e-04, 9.8947203e-05, 0.0000000e+00,
        1.1193836e-02],
       [5.4722326e-04, 4.5481557e-04, 5.5663171e-04, 4.5626750e-04,
        6.3341926e-03],
       [6.3093100e-04, 6.2016491e-04, 6.5473723e-04, 6.4431550e-04,
        6.8652160e-03],
       ...,
       [6.7185313e-01, 6.7325997e-01, 6.7880189e-01, 6.7853564e-01,
        1.1325711e-01],
       [7.0060766e-01, 7.0091999e-01, 7.0047134e-01, 6.9820404e-01,
        1.0618710e-01],
       [7.1808356e-01, 7.2158784e-01, 7.2548938e-01, 7.2342026e-01,
        1.3153352e-01]], dtype=float32)
```

Step 2: Supervised Learning Transformation

```python
# Converting time series data into the format of a supervised learning problem
def series_to_supervised(data, list_feature, n_in, n_out, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    # Used to store feature columns and corresponding names
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [(list_feature[j]+'(t-%d)' % (i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [(list_feature[j]+'(t)') for j in range(n_vars)]
        else:
            names += [(list_feature[j]+'(t+%d)' % (i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)

    return agg

n_in = 3  #Use features from the last number of days as input
n_out = 2  #Forecast prices for the next few days
# frame as supervised learning
reframed = series_to_supervised(scaled, list_feature, n_in, n_out)
```

Time Series -> Supervised Learning

- To train models that predict future outcomes based on historical data
- Generates **lagged features** to represent **past observations**
- Generate **targets** to represent f**uture price movements**
- Rows containing missing values are removed

	close(t-3)	high(t-3)	low(t-3)	open(t-3)	NVT(t-3)	close(t-2)	high(t-2)	low(t-2)	open(t-2)	NVT(t-2)	...
3	0.000245	0.000145	0.000099	0.000000	0.018329	0.000547	0.000455	0.000557	0.000456	0.010371	...
4	0.000547	0.000455	0.000557	0.000456	0.010371	0.000631	0.000620	0.000655	0.000644	0.011241	...
5	0.000631	0.000620	0.000655	0.000644	0.011241	0.000442	0.000383	0.000455	0.000391	0.020952	...
6	0.000442	0.000383	0.000455	0.000391	0.020952	0.000546	0.000445	0.000558	0.000458	0.025562	...
7	0.000546	0.000445	0.000558	0.000458	0.025562	0.000706	0.000619	0.000728	0.000643	0.012613	...
...	...	...	...	...	...	...	...	...	...	...	...
2551	0.615244	0.616486	0.615865	0.613859	0.349655	0.615002	0.618918	0.621808	0.623769	0.349655	...
2552	0.615002	0.618918	0.621808	0.623769	0.349655	0.604987	0.606381	0.608755	0.609593	0.349655	...
2553	0.604987	0.606381	0.608755	0.609593	0.349655	0.619526	0.619602	0.625160	0.623163	0.349655	...
2554	0.619526	0.619602	0.625160	0.623163	0.349655	0.644377	0.644853	0.649146	0.647955	0.349655	...
2555	0.644377	0.644853	0.649146	0.647955	0.349655	0.658583	0.659225	0.664237	0.664356	0.349655	...

2553 rows × 25 columns

Step 3: Input and Output Matrix Split

```
list_use_feature = []
for j in range(len(list_feature)):
    for i in range(n_in):
        list_use_feature.append(list_feature[j]+'(t-%d)' % (i+1))

X = reframed[list_use_feature]
print('X columns: ',X.columns)

list_use_result = []
list_use_result.append('close(t)')
for i in range(n_out-1):
    list_use_result.append('close(t+%d)' % (i+1))
Y = reframed[list_use_result]
print('Y column: ', Y.columns)
```

```
X columns:  Index(['close(t-1)', 'close(t-2)', 'close(t-3)', 'high(t-1)', 'high(t-2)',
       'high(t-3)', 'low(t-1)', 'low(t-2)', 'low(t-3)', 'open(t-1)',
       'open(t-2)', 'open(t-3)', 'NVT(t-1)', 'NVT(t-2)', 'NVT(t-3)'],
      dtype='object')
Y column:  Index(['close(t)', 'close(t+1)'], dtype='object')
```

Input **X**

Historical characteristics from **close(t-n_in) to close(t-1)** for predicting future price movements

Output **Y**

The target value that the model will try to predict, focusing on the **close(t) to close(t+(n_out-1))**

Step 4: Train-Test Split

```
X = X.values
Y = Y.values
n_train_days = int(X.shape[0]/3)
print('Number of samples in the training set: ', n_train_days)

# The first n_train_days samples are used as the training set
# and the remaining samples are used as the test set
train_X, train_Y = X[:n_train_days, :], Y[:n_train_days, :]
test_X, test_Y = X[n_train_days:, :], Y[n_train_days:, :]

# reshape input to be 3D [batchs, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_Y.shape, test_X.shape, test_Y.shape)
```

```
Number of samples in the training set:  851
(851, 1, 15) (851, 2) (1702, 1, 15) (1702, 2)
```

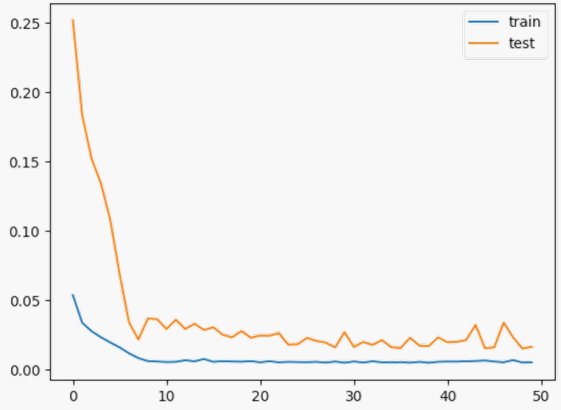
Reshape to 3D

- The data is divided into a training set and a test set, the first "**n_train_days**" samples are assigned to the training set and the rest of the samples form the test set
- Reshape the input features into a 3D tensor format suitable for neural network input, including dimensions such as **batch size**, **time step** and **number of features**

Step 5: Building and Training the LSTM Model

```
# design network
model = Sequential()   # Create a sequential model, i.e. a linear cascade model
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(train_Y.shape[1]))  # full connectivity layer
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_Y, epochs=50, batch_size=72, validation_data=(test_X, test_Y), verbose=2, shuffle=False)
```

LSTM Layer	Processing of sequence data
Dense Layer	Temporal patterns are transformed into final predictions
Loss	Mean Absolute Error (MAE)
Optimization	Adam
Fitting	Epoch: 50, Batch Size: 72, Verbose: 2



Step 6: Model Evaluation

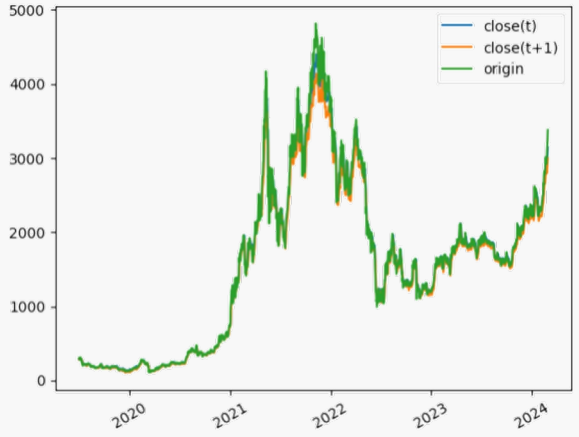
```
# Prediction of test set data using trained model
Yhat = model.predict(test_X)
inv_Yhat0 = concatenate((scaled[n_in:-n_out+1, :3][n_train_days:,:],
                         Yhat[:,:1],
                         scaled[n_in:-n_out+1, 4:][n_train_days:,:]), axis=1)
inv_Yhat0 = scaler.inverse_transform(inv_Yhat0)
inv_Yhat0 = inv_Yhat0[:,3]

inv_Yhat1 = concatenate((scaled[n_in:-n_out+1, :3][n_train_days:,:],
                         Yhat[:,1:],
                         scaled[n_in:-n_out+1, 4:][n_train_days:,:]), axis=1)
inv_Yhat1 = scaler.inverse_transform(inv_Yhat1)
inv_Yhat1 = inv_Yhat1[:,3]
```

Predicted close(t) & close(t+1)

```
array([[0.05937247, 0.05573027],
       [0.0604595 , 0.05700527],
       [0.05925619, 0.05645518],
       ...,
       [0.6466017 , 0.6074855 ],
       [0.663042  , 0.62323713],
       [0.6799099 , 0.63872886]], dtype=float32)
```

```
predictiony0 = inv_Yhat0.reshape(-1)
predictiony1 = inv_Yhat1.reshape(-1)
originaly = values['close'].values[n_in:-n_out+1][n_train_days:]
```

predicted closing price & actual Ether price



05

Strategy & Back-testing

LIANG Haojin

Strategy

```
dataset = {'date':x,
           'close(t-1)':originaly,
       'close(t)':predictiony0,
       'close(t+1)':predictiony1}
df = DataFrame(dataset)
# df.set_index(['date'], inplace = True, drop=True)
df['close(t+1)-close(t-1)'] = df['close(t+1)'] - df['close(t-1)']
df['isbuy'] = df['close(t+1)-close(t-1)'] > 0
df['issell'] = df['isbuy'].shift(2) #n_out=2
df['issell'].fillna(value= False, inplace=True)
```

date	close(t-1)	close(t)	close(t+1)	close(t+1)-close(t-1)	isbuy	issell
2019-07-03	303.11823	303.60574	296.81590	-6.30233	False	False
2019-07-04	284.41977	309.02450	303.19293	18.77316	True	False
2019-07-05	287.97790	304.57602	299.69812	11.72022	True	False
2019-07-06	287.50458	303.37424	295.83423	8.32965	True	True
2019-07-07	305.76273	296.10210	288.60770	-17.15503	False	True

Two-days cycle

close(t-1): current price
close(t): predicted current price
close(t+1): predicted tomorrow price

Signal generation

close(t-1)<close(t+1) → buy signal
close(t-1)>close(t+1) → sell signal

Strategy objective

Minimize maximum drawdowns
Conservative with lower return

Strategy

```
initlocation = 0 #初始仓位为0
initUS = 2 #初始美元现金为2
maxlocation = 2 #满仓为2
def function(isbuy, issell):
    if issell and (not isbuy): return -1
    if isbuy and issell: return 0
    if (not isbuy) and (not issell): return 0
    if isbuy and (not issell): return 1

df['location action'] = df.apply(lambda x: function(x.isbuy, x.issell), axis = 1)
df['location status'] = df['location action'].expanding().sum()
df['location status'] = df['location status'] / df['location status'].max()
df
```

close(t+1)-close(t-1)	isbuy	issell	location action	location status
-6.30233	False	False	0	0.0
18.77316	True	False	1	0.5
11.72022	True	False	1	1.0
8.32965	True	True	0	1.0
-17.15503	False	True	-1	0.5

Position calculation

position	Isbuy=True	Isbuy=False
Issell=True	0	-1
issell=False	1	0

(Max position=2)
Location action: cumulative position
Location status: normalize above

Back-testing

```
df['cycle growth rate'] = df['close(t-1)'].diff()
df['cycle growth rate'] = df['cycle growth rate'].shift(-1)
df['cycle growth rate'].fillna(value=0, inplace=True)
df['cycle growth rate'] = df['cycle growth rate'] /  df['close(t-1)']
df['Asset cycle growth rate'] = df['location status'] * df['cycle growth rate']
df['cycle growth rate'] = df['cycle growth rate'].shift(1)
df['cycle growth rate'].fillna(value=0, inplace=True)
df['Asset cycle growth rate'] = df['Asset cycle growth rate'].shift(1)
df['Asset cycle growth rate'].fillna(value=0, inplace=True)
df['Asset cycle growth rate'] = df['Asset cycle growth rate'] + 1
df['Asset growth rate'] = df['Asset cycle growth rate'].cumprod()
df['ETH growth rate'] = (df['cycle growth rate']+1).cumprod()

plt.figure()
plt.plot(df.index,df['ETH growth rate'].values.reshape(-1), 'r', label='ETH growth rate')
plt.plot(df.index,df['Asset growth rate'].values.reshape(-1), 'b', label='Asset growth rate')
plt.xticks(df.index[::100],  rotation='vertical')

plt.legend()
plt.show()
print(df['ETH growth rate'].tail())
print(df['Asset growth rate'].tail())
```

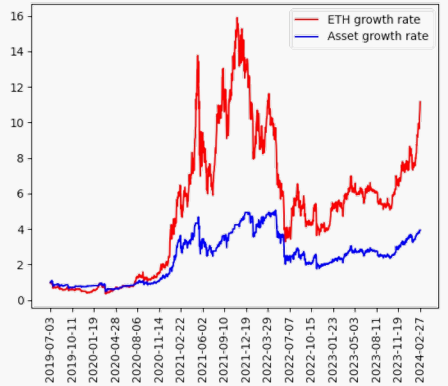
Calculate performance curve

Cycle growth rate
Asset growth rate
ETH growth rate

Plot performance curve

ETH price vs strategy

position	Return
ETH	11.155
Strategy	3.93



06

Results

HUANG Shiqi

Results of Model Comparison

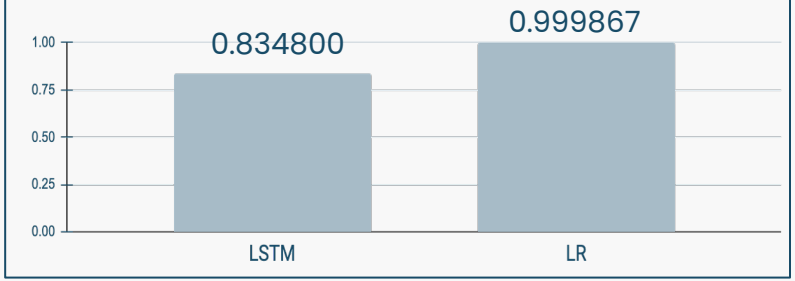
Mean Square Error

LSTM	0.004739
LR	155.519430

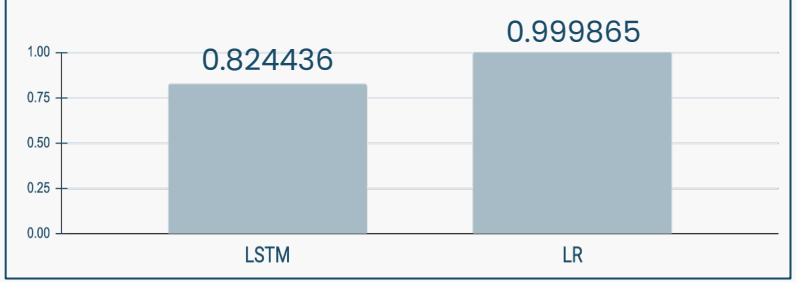
Root Mean Square Error

LSTM	0.068838
LR	12.470743

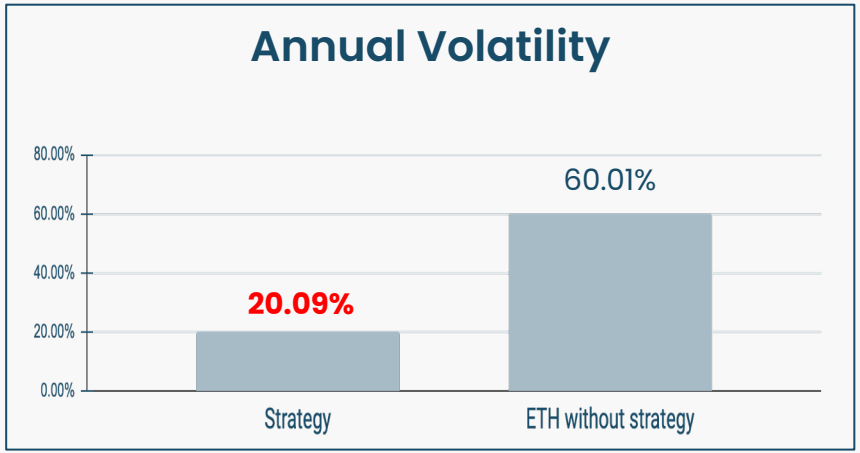
Explained Variance Ratio

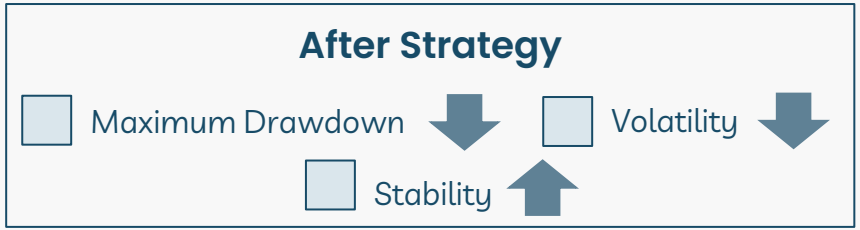


R-squared



Performance Analysis of Strategy





Measures	Strategy	Without Strategy
Holding Period Return	293%	1015.47%
Annual Return	2.73%	9.48%
Sharpe Ratio	1.89	1.38
Maximum Drawdown	0.65	0.66
Calmar Ratio	4.19	14.30

07

Summary

CHEN Yunfan

Recap of key findings and insights

Superior Performance of LSTM

The LSTM model outperformed the Linear Regression (LR) model in terms of predictive accuracy, as evidenced by significantly lower Mean Square Error (MSE) and Root Mean Square Error (RMSE).

Profitability of ML-Based Trading Strategies

The backtesting of the ML model demonstrated its potential in generating profitable trading strategies. The analysis revealed a substantial holding period return of 293.00% and an annual return of 4.86%.

Further Considerations

Multi-Factor Analysis

Incorporating market sentiment, social media trends, trading indicators, and macroeconomic variables enhances predictive power.

Hyperparameter Optimization

Fine-tuning parameters like LSTM layers, hidden units, learning rate, and dropout rate is crucial for model performance.

Trading Cost

Accounting for transaction fees, slippage, and other costs provides a realistic assessment of strategy performance in real-world trading scenarios.

However, it is important to note that despite the inclusion of additional factors and careful hyperparameter tuning, the predictive accuracy of LSTM models for cryptocurrency returns still exhibits inherent stochasticity.

Thanks!

Group 6

FTEC 5530

Evaluating the Effectiveness of Machine Learning in Cryptocurrency Trading