پیش بینی بیت کوین

پژوهشگر:مهدی عاقلی آشان

استاد:دکتر خطایی

دانشگاه آزاد اسلامی واحد مراغه

بهار ۱۴۰۴

۱.مقدمه:

در این پروژه سعی بر این شده تا با استفاده از یک مدل از شبکه های عصبی LSTM ، قیمت بیت کوین پیش بینی گردد. بازار رمز ارز ها به خصوص بیت کوین دارای پیچیدگی های رفتاری غیر خطی و نوسانات شدید است ؛بنابراین مدل های سنتی یادگیری ماشین توانایی و دقت لازم برای روند قیمت و بازار را ندارند.در این مدل پیشنهادی, ابتدا داده های قیمت روزانه بیت کوین همراه با اندیکاتور های فنی(که از این داده ها استخراج شدند)به عنوان ورودی به شبکه LSTMداده میشود تا وابستگی های زمانی بلند مدت داده هارا استخراج کند و توانایی پیش بینی قیمت روز بعد را داشته باشد. در این پروژه از شبکه LSTM خالص استفاده شد با تنظیم دقیق پارامتر در تست و ازمون مقدار تست لاست به طرز چشمگیری کاهش پیدا کرد که نشان از بهبود دقت مدل در استخراج الگوهای قیمتی و تصمیم گیری جهت دار در بازار بیت کوین است.این پروژه میتواند به عنوان گامی موثر در در طراحی سیستم های هوشمند تحلیلگر بازار رمز ارز ها مورد استفاده قرار گیرد.

۲.پیش زمینه:

در این بخش سعی بر این شده تا مفاهیم پایه ای که در این پروژه و توسعه مدل پیشنهادی استفاده شده است؛معرفی گردد تا افراد و خوانندگانی که آشنایی اولیه با این مفاهیم ندارند نیز بتوانند روند انجام پروژه را دنبال کنند.

۲.۱ شبکه عصبی LSTM:

Long short-term memory(LSTM) نوع خاصی از شبکه عصبی بازگشتی است که برای مدیریت و پیش بینی موثر توالی داده ها و حل مشکلاتی مانند ناپدید شدن گرادیان طراحی شده است. LSTM ها شامل سلول هایی از نوع حافظه هستند که اطلاعات را در طول زمان ذخیره میکنند و کاری میکند تا شبکه قادر به انعکاس ورودی های گذشته باشد.

این معماری شامل سه نوع دروازه است: دروازه ورودی، خروجی و فراموشی که جریان اطلاعات را به داخل و خارج از سلول های حافظه تنظیم میکند و به شبکه اجازه میدهد یاد بگیرد کدام اطلاعات را حفظ یا حذف کند.

(Okut,2021),(Rokui & Adachi,2022)

در شبکه های عصبی بازگشتی مشکلاتی همچون ناپدید شدن گرادیان و انفجار گرادیان وجود دارد که اغلب مرتبط به RNNها هستند و مانع یادگیری وابستگی های بلند مدت می شوند.LSTMها با استفاده از با استفاده رویکرد پسا انتشار مبتنی بر گرادیان نزولی، میتوانند داده ها را در بازه بلند مدت بهتر مدیریت کرده و معادلات غیر خطی را حل کنند و بهتر از شبکه های عصبی سنتی عمل کنند

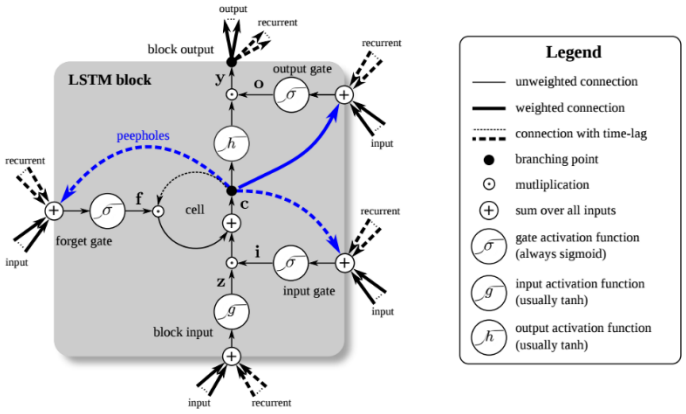
(Sahu et al.,2023)

تحقیقات نشان داده است که آنها در پیش بینی داده های سری زمانی فصلی بهتر از سایر مدل ها عمل میکنند و آنهارا در زمینه هایی که نیاز به پیش بینی زمانی دقیق دارند،ارزشمند میکنند.

(Wang et al.,2019)("Cell-expanded Long Short-term Memory",2022)

در حالیکه LSTMها ابزار قدرتمندی برای پیش بینی توالی ها به خصوص توالی های بلند مدت هستند، اما بدون محدودیت نیستند. پیچیدگی معماری انها میتواند منجر به افزایش محاسبات و در برخی موارد یک سلول حافظه ممکن است به طور کامل پیچیدگی های داده های سری زمانی را ثبت نکند.محققان بهبود هایی مانند گنجاندن سلول های حافظه اضافی برای بهبود عملکرد در چنین سناریو هایی مطرح کرده اند.

(Rokui &Adaci ,2022)



شکل ۱ . معماری شبکه عصبی LSTM

۲.۲ داده های مورد نیاز برای پیش بینی:

OHCLV:

Open: اولین قیمت که دارای در شروع یک بازه زمانی معامله شده ،نقطه شروع است و با مقایسه close میتوان صعودی یا نزولی بودن کندل را تشخیص داد.

High: برای تشخیص مقاومت ها یا بالاترین سطحی که قیمت در آن بازه توانسته به آن برسد.

Low: کمترین قیمت در بازه معامله شده برای تشخیص حمایت یا پایین ترین سطحی که قیمت آن را لمس کرده است.

Close: آخرین قیمت معامله شده در پایان بازه زمانی و مهمترین قیمت در تحلیل تکنیکال.

Volume: تعداد قرارداد هایی که در اون بازه معامله شده و در تشخیص قدرت حرکت قیمت مثلا اگر قیمت افزایش یافته باشد ولی volume پایین باشد ممکن است حرکت ضعیف باشد.

Quote asset volume : حجم معاملات بر حسب ارز یعنی ارزش دلاری کل معاملات در آن کندل یا به عبارت دیگر بررسی قدرت نقدینگی بازار در آن کندل.

Number of trade: تعداد کل تراکنش ها یا معاملات انجام شده در آن بازه زمانی؛ اگر حجم بالا باشد ولی معامله کم یعنی معاملات بزرگ انجام شده و بالعکس یعنی معاملات خرد زیاد انجام شده است. برای تشخیص تحت کنترل بودن بازار توسط معامله گر های بزرگ یا کوچک استفاده میشود؛ همچنین برای تشخیص ریزش یا رشد ناگهانی ناشی از اسپایک های فعالیت استفاده میشود.

Taker buy base asset volume: مقدار دارایی های پایه که توسط خریداران خریداری شده است اگر مقدار زیادی از حجم توسط takerها خریداری شده باشد یعنی قدرت خریدار بالا است.

اندیکاتور های حرفه ای:

اندیکاتور های حرفه ای ابزار هایی هستند که با استفاده از فرمول های های ریاضی و داده های قیمت و حجم به معامله گر ها کمک میکنند تا روند بازار، نقاط ورود و خروج مناسب و قدرت ضعف روند را تشخیص دهند.

SMA-14: میانگین متحرک ساده ۱۴ دوره ای یعنی میانگین قیمت بسته شدن در ۱۴ مندل قبلی؛کاربرد آن در فیلتر کردن نوسانات کوتاه مدت و تعیین جهت روند کلی ، پشتیبانی برای سایر اندیکاتور ها مثل Bollinger brand

از فرمول زیر بدست می آید:

$$SMA14(t) = 141i = 0\sum 13Pt - i$$

$$SMA14(t) = 14Pt + Pt - 1 + Pt - 2 + \cdots + Pt - 13$$

که در آنpt معمولا قیمت پایانی روز tام است.

RSI-14: شاخص قدرت نسبی که از فرمول زیر بدست می آید:

$$RS=\frac{avrage\ gaib}{avrage\ loss}$$

$$RSI = \left(\frac{100}{RS + 1}\right) - 100$$

Average gain: میانگین افزایش قیمت در ۱۴ دوره گذشته.

Average loss: میانگین کاهش قیمت (قدر مطلق) در ۱۴ دوره گذشته اگر بالای ۷۰ باشد یعنی خرید بیش از حد؛ اگر کمتر از ۳۰ باشد یعنی فروش بیش از حد.

MACD: به شناسایی جهت ،قدرت و مومنتوم روند کمک میکند این اندیکاتور بر اساس میانگین متحرک نمایی (EMA) ساخته شده که شامل اجزای خط ، خط سیگنال، و هیستوگرام است که از فرمول های زیر بدست می اید:

$$MACD = EMA26 - EMA12$$

$$signal = EMA9(MACD(t))$$

$$histogram = signal - MACD$$

Bollinger bands: توسط جان بولینگر ابداع شده و برای اندازه گیری نوسانات بازار و شناسایی نقاط احتمالی برگشت قیمت استفاده میشود.

Upper band: برابر است با باند میانی به علاوه چند برابر انحراف معیار قیمت ها، معمولا ضریب دو استفاده میشود.

$$upper\ band = \sigma * k + SMA(n)$$

Lower band: برابر است با باند میانی منهای چند برابر انحراف معیار،معمولا از ضریب دو استفاده میشود.

$$lowe\ band = \sigma * k - SMA(n)$$

۲.۳ مقیاس بندی:

robust scaler:

یک روش استاندارد سازی داده ها است که مانند standard scaler نیست که از میانگین و انحراف استاندارد استفاده کند، بلکه به جای آن از میانه و دامنه میان چارکی استفاده میکند و هدف ان کاهش تاثیر داده های پرت و نوسان ها در مقیاس بندی است ، چون میانه و IQR نسب به مقادیر پرت مقاوم تر هستند در داده های سری زمانی به خصوص رمز ارز ها نوسان ها زیاد هستند و نمیتوان به عنوان داده پرت در نظر گرفت چون نوسان ها ویژگی های اصلی بازار هستند.

از فرمول زیر بدست می آید:

$$IQR(X) = Q1 - Q3$$

$$x' = \frac{x - median(x)}{IQR(X)}$$

Median(x): میانه داده های x

Q1: چارک اول (۲۵مین درصد داده ها)

Q3: چارک سوم (۷۵امین درصد داده ها)

۲.۴ توابع فعال سازی:

توابع فعال سازی در معماری خود LSTM نقش حیاتی دارند زیرا تعیین میکنند که چگونه اطلاعات بین سلول های حافظه در جریان باشد چه داده ای اهمیت دارد و تصمیم بگیرد چه چیز هایی را حفظ یا فراموش کند.

Sigmoid : این تابع در گیت فراموشی،ورودی و خروجی استفاده میشود چون یک خروجی در بازه [0,1] میدهد اگر صفر باشد چون ضرب میشود اطلاعات فراموش و عبور داده رخ نمیدهد و اگر یک باشد یعنی باید کاملا عبور بدهد اگر خروجی بین صفر و یک باشد هرچقد به یک نزدیک تر باشد یعنی اطلاعات اهمیت بیشتری دارند و حفظ میشوند.

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

Hyperbolic tangent(tanh) : تابع فعال سازی تانژانت هذلولوی با تابع فعال سازی سیگموئید بسیار شباهت دارد در این شبکه از آن برای ایجاد مقادیر کاندید برای ذخیره در خافظه و به روزرسانی حالت سلولی همچنین برای تولید خروجی نهایی استفاده میشود زیرا بین خروجی در منفی یک و یک ایجاد میکند که کمک به نگه داری پایداری گرادریان در مقایسه ***ReLU*** دارد و مناسب داده های توزیع شده حول صفر پس دامنه وسیع تری نسبت به سیگموئید دارد که کمک به یادگیری غنی تر میشود.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Leaky ReLU: پس از اینکه LSTM توالی را پردازش میکند ، خروجی آن وارد یک یک یا چند لایه fully connected میشود تا خروجی LSTM به فضای قابل تفسیر مثلا احتمال ،عدد یا کلاس تبدیل شود و یا برای فشرده سازی یا گسترش بعد ویژگی ها؛ در بین این لایه ها ما نیازمند یک تابع فعال سازی هستیم یکی از این تابع ها Leaky ReLU است چون این تابع بر خلاف ReLU معمولی که برای مقادیر منفی صفر خروجی میدهد باعث

میشود اگر نورونی وارد این ناحیه شود و مقدا گرادیان دریافت نکند ممکن است هیچ وقت دوباره فعال نشود یا به اصطلاح مرگ نورون اتفاق بیوفتد اما Leaky ReLU مقدار کمی گرادیان عبور میدهد تا این را کاهش دهد.

$$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x < 0 \end{cases}$$

الفا یک مقدار کوچک مثبت است است مثلا *0.01*

۲.۵ بهینه ساز:

شبکه های عصبی بازگشتی مانند LSTM همانطور که قبلا اشاره کردیم مشکل فراموشی گرادیان و یا انفجار گرادیان دارند بنابراین بهینه سازی سختتری نسبت به شبکه های عصبی feedforwardیا پیش ران دارند.

RMSprop یک الگوریتم بهینه سازی مبتنی بر گرادیان است که با استفاده از مانگین نمایی متحرک از مربع گرادیان؛نرخ یادگیری را برای هر پارامتر تنظیم میکند؛ هدف آن جلوگیری از نوسانات زیاد در بروزرسانی ها به خصوص در شبکه های بازگشتی و سرعت دادن به همگرایی رسیدن مدل است. در این الگوریتم گرادیان با رباطه زیر اپدیت میشود:

$$G_i(t) = \beta G_i(t-1) + (1-\beta) \left|\frac{\partial L}{\partial w_i}(t)\right|^2$$

$$; G_i(0) = 0$$

بعدا یک الگوریتم قدرتمند دیگری با تلفیق ایده های تکانه و تغییر نرخ یادگیری بر اساس گرادیان ایجاد شد که Adam نام گرفت،در این الگوریتم اثر تکانه به رابطه RMSprop اضافه می شود.

$$RMSprop: \begin{cases} \begin{cases} w_i(t+1) = w_i(t) - lr \left|\frac{\partial L}{\partial w_i}(t)\right|^2 \\ ; lr = -\frac{lr_0}{\sqrt{G_i(t)} + e} \end{cases} \\ \begin{cases} G_i(t) = \beta G_i(t-1) + (1-\beta) \left|\frac{\partial L}{\partial w_i}(t)\right|^2 \\ ; G_i(0) = 0 \end{cases} \end{cases}$$

$$momentum: w_i(t+1) = w_i(t) - lr \left|\frac{\partial l}{\partial w_i}(t) + \alpha \frac{\partial l}{\partial w_i}(t-1)\right|$$

بدین ترتیب پس از اعمال پاره ای از تبدیلات رابطه اپدیت وزن ها در الگوریتم Adam به شکل زیر در می آید:

$$w_i(t+1) = w_i(t) - \frac{lr_0}{\sqrt{G_i(t)} + e} M_i(t)$$

$$M_i(t) = \alpha M_i(t-1) + (1-\alpha) \frac{\partial l}{\partial w_i}(t)$$

$$; M_i(0) = 0$$

۲.۶ تابع هزینه:

در پیش بینی های سری زمانی با lstmمثلا همین پروژه پیش بینی قیمت بیت کوین استفاده از SmoothL1 به جای MSE میتواند مفید باشد به خصوص وقتی داده ها ناپایدار و پرنوسان مانند بیت کوین داشته باشیم؛تابع هزینه SmoothL1ترکیبی از MSE,MAE است.این تابع برای مسائلی که دارای داده های پرت یا پرنوسان هستند بسیار مناسب است همچنین استفاده از این تابع در مدل LSTM برای پیش بینی مقادیر عددی میتواند پایداری مدل را افزایش دهد. از فرمول زیر بدست می آید:

$$SmoothL1(x,y) = \begin{cases} 0.5(x-y)^2, & if|x-y| < 1 \\ \delta(|x-y| - 0.5.\delta), & otherwise) \end{cases}$$

X: مقدار پیش بینی شده

Y: مقدار واقعی

δ: آستانه مثلا یک، تنظیم کننده تغییر رفتار بین MSE, MAE

۲.۷کتابخانه های استفاده شده:

این پروژه با زبان برنامه نویسی پایتون توسعه یافته، در این پروژه از فریمورک pytorch و از کتابخانه های joblib،matplotlib،os،sklearn،pandas،numpy requests،pandas_ta، استفاده شده است.

۳.مراحل انجام پروژه:

معمولا در پروژه های یادگیری ما سه مرحله اصلی داریم که باید به ترتیب آن هارا انجام دهیم:۱.پیش پردازش داده ها که شامل جمع آوری دیتاست، بررسی داده ها،شناسایی داده های پرت و انجام عملیات برای جایگذاری انها مثل استفاده از ماینگین،مدیریت داده های گمشده و انجام عملیات روی آنها مثل حذف یا جایگزین کردن آنها با میانگین،در صورت وجود داده های str تبدیل انها به تنسور به وسیله OHDتا به توان آن هارا به مدل شناساند،در صورت وجود داده های لیبل به صورت str تبدیل آنها به عدد که بیشتر در مسائل طبقه بندی این مورد مشاهده میشود، تقسیم دیتاست به داده های آموزش تست و ارزیابی، مقیاس بندی داده ها به منظور جلوگیری از تسلط برخی ویژگی ها بر سایر ویژگی ها،تبدیل داده ها به حالت تنسور برای ورودی مدل، استفاده از دیتالودر و مینی بچ برای تقسیم داده ها به گروه کوچک تا تا مدل سریع تر اموزش دیده و حافظه کمتر مصرف گردد. ۲.اموزش مدل که شامل تعیین مسئله مثل رگرسیون و انتخاب الگوریتم یادگیری مناسب و یا شبکه عصبی مناسب برای اموزش، آموزش مدل و بررسی عملکرد مدل به وسیله تست لاست یا mape ۳.پیش بینی که شامل پیش بینی مثلا قیمت با ویژگی هایی که بهش میدهیم.همانطور که اشاره کردم در این پروژه هم مانند سایر پروژه های یادگیری سه مرحله اصلی توسعه یافته به اضافه یک بخش دیگر هم اضافه شده به نام فاین تیونینگ، فاین تیونینگ در پروژه هایی مثل سری زمانی اهمیت بسیار مهمی دارد در این روش مدل پس از اموزش قیمت فردای دیتاست را پیش بینی میکند در حالیکه قیمت بیت کوین در هر لحظه در حال تغییر هست و ما نیازمند هستیم تا وزن و بایاس مدل خود را اپدیت نگه داریم تا برای پیش بینی های بعدی مورد استفاده گردد در مورد نحوه پیاده سازی در بخش فاین تیونینگ توضیحات کاملی ارائه خواهد شد؛پروژه به صورت ماژولار است مرحل اصلی به صورت ماژول و سایر بخش ها به صورت توابع پیاده سازی شده است.

۳.۱ ماژول DataPreproccesing:

اولین گام برای توسعه یک مدل یادگیری پیش پردازش داده ها است،این مرحله مهم ترین مرحله یادگیری هست چون اگر داده ها دارای پرت و یا نوسان های غیرمتعارف باشند مدل به درستی تعمیم نمی پذیرد و یا اگر داده ها درست تقسیم و سازگار با ورودی مدل نداشته باشند مدل کار نکرده و پروژه به باگ خواهد خورد. این ماژول شامل توابع:

Out, andicator,seq,seq_prid, PreDate است.

۳.۱.۱ جمع آوری دیتاست:

برای اموزش مدل ما نیازمند داده های مناسب و معتبر هستیم دیتاست این پروژه از طریق صرافی بایننس به صورت سری زمانی و روزانه جمع آوری شده که شامل OHCLV است که در بخش ۲.۲ کاملا توضیح داده شده که این داده ها چی هستند.داده ها از تاریخ 2017/08/17 تا 2025/03/21 است و شامل ۲۷۷۴ وکتور است.

۳.۱.۲ امپورت های این ماژول:

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas_ta as ta
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
import torch
from torch.utils.data import DataLoader,TensorDataset
import joblib
```

شکل۲.کتابخانه های استفاده شده در ماژول DataPreprocesing

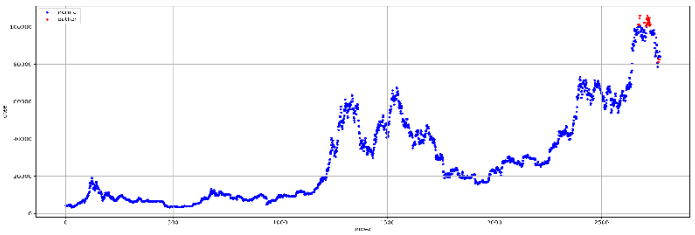
۳.۱.۳ تابع out :

با بررسی های انجام شده و رسم نمودار ها مشخص شد که دیتاست دارای داده های پرت نیست و نوسان های موجود ویژگی های اصلی بازار هستند ولی در ویژگی close time داده های غیرمتعارف مانند ۱۷۴۲۶۰۱۵۹۹۹۹۹۹۹ مشاهده شد که با تبدیل آن به تاریخ برابر سال ۵۶۷۶ شد که عملا داده اشتباه است که باعث ایجاد مشکلی همچون تغییر رنج داده در بخش تست شد که در بخش آتی راهکار درست کرد آن ارائه شده است. در این تابع برای تشخیص داده های پرت وناهنجار از الگوریتم یادگیری ماشین به نام isolation forest استفاده شده این الگوریتم بر خلاف بسیاری از الگوریتم ها که سعی در مدل سازی داده های نرمال برای شناسایی ناهنجاری استفاده میکنند،این الگوریتم سعی دارد داده های ناهنجار را ایزوله کند چون این داده ها راحتر و سریع تر از سایر نمونه ها قابلیت ایزوله شدن دارند. این الگوریتم با ساخت درخت های تصادفی کار میکند؛ در هر درخت داده ها به صورت تصادفی تقسیم میشوند این تقسیم ها ادامه می یابد تا هر نمونه ایزوله گردد؛ نمونه هایی که زود ایزوله میشوند یعنی عمق کمتری دارند احتمالا ناهنجار هستند. این الگوریتم سریع و بسیار سبک برای دیتاست های بزرگ هست و چون unsupervised است نیازی به لیبل ندارد و عملکرد بهینه ای در فضا با ابعاد بالا دارد تصویر زیر شامل پیاده سازی این تابع و رسم پلات انها است در ادامه نمودار های خروجی آورده شده است.

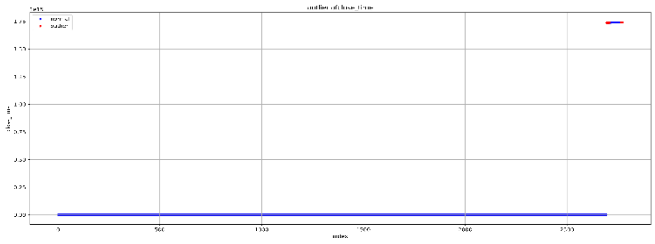
```python
def out():
    '''شناسایی داده های پرت با IsolationForest'''
    dataset=pd.read_csv("BTCUSDT_combine.csv")
    os.makedirs("Scatter_out",exist_ok=True)
    for c in dataset.columns:
        dataset[c]=dataset[c].astype(float)
        x=dataset[c].values.reshape(1,-1)
        model=IsolationForest(contamination=0.01,random_state=42)
        y=model.fit_predict(x)
        plt.figure(figsize=(20,8))
        plt.scatter(range(len(x)),x[:,0],c="blue",label="normal",s=10)
        outlier=[i for i in range(len(y)) if y[i]==-1]
        plt.scatter(outlier,x[outlier],c="red",label="outlier",s=10)
        plt.xlabel("index")
        plt.title(f"outlier of{c}")
        plt.ylabel(f'{c}')
        plt.legend()
        plt.grid(True)
        img=os.path.join("Scatter_out",f"{c}_outliers.png")
        plt.savefig(img,dpi=600)
        plt.close()
out()
```

شکل۳ .کد پیاده سازی شده out

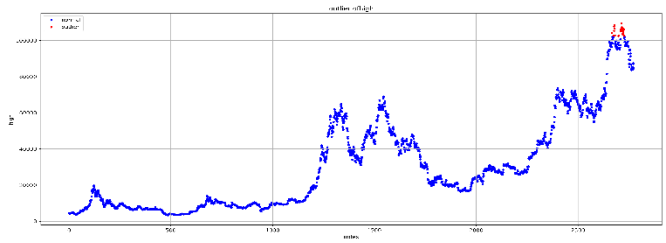
نمودار برخی ویژگی های مهم رسم شده:



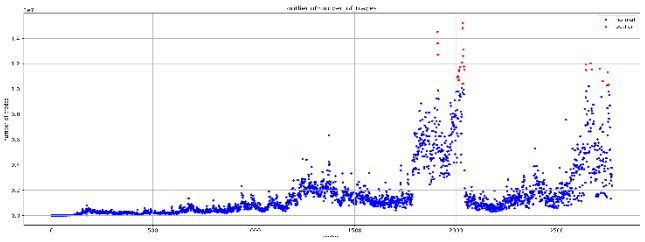
شکل۴ .نمودار ستون close



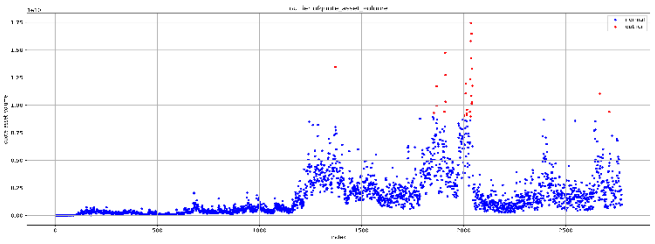
شکل۵ . نمودار ستون close time که ناهنجاری در آن دیده میشود



شکل۶ . نمودار ستون open در نگاه اول شبیه close است چون قیمت پایانی روز قبل قیمت شروع روز بعد است.



شکل ۷ . نمودار ستون number of trade که شامل ناهنجاری است ولی با بررسی که انجام شد مشخص گردید انها ویژگی بازار هستند در آنروز ها به خاطر عواملی تعداد ترید زیاد بوده



شکل۸ . نمودار ستون quote asset volume

۳.۱.۴ تابع andicator :

در بخش ۲.۲ اهمیت و نحوه محاسبه اندیکاتور های حرفه ای را دیدیم اینها ابزار های مهمی برای تحلیل، تحلیل گر ها هستند و هدف ما ازآموزش مدل این است که مدل مانند یک تحلیلگر واقعی رفتار بکند پس باید این اندیکاتور های حرفه ای را محاسبه و دیتاست خود را اپدیت کرده تا مدل الگو ها را کشف کند برای محاسبه این اندیکاتورها از کتابخانه pandas-ta استفاده میکنیم ابتدا باید مشخص کنیم که داده های ما فاقد داده های nullباشد چون اگر داده گمشده در دیتاست وجود داشته باشد به باگ برخورد میکنیم و نمیتوان محاسبات لازمه را انجام داد خروجی بررسی داده های null به شرح زیر است:

```
'''number of null in timestamp: 0
      number of null in open: 0
      number of null in high: 0
      number of null in low: 0
      number of null in close: 0
      number of null in volume: 0
      number of null in close_time: 0
      number of null in quote_asset_volume: 0
      number of null in number_of_trades: 0
      number of null in taker_buy_base: 0
      number of null in taker_buy_quote: 0
      number of null in ignore: 0'''
```

شکل ۹ . خروجی بررسی تعداد داده های null

تصویر زیر کد پیاده سازی این تابع است:

```
def andicator(dataset):
    '''بررسی داده های گمشده و افزودن اندیکاتور های فنی'''
    for i in dataset.columns:
        miss=dataset[i].isnull().sum()
        print(f"number of null in {i}: {miss}")
    dataset['SMA_14']=dataset['close'].rolling(window=14).mean()
    dataset['RSI_14']=ta.rsi(dataset['close'],length=14)
    macd=ta.macd(dataset['close'])
    dataset['MACD']=macd['MACDs_12_26_9']
    dataset['MACD-signal']=macd['MACDs_12_26_9']
    dataset["MACD-hist"]=macd['MACDs_12_26_9']
    bollinger=ta.bbands(dataset['close'],length=20)
    dataset['BB_upper']=bollinger['BBU_20_2.0']
    dataset['BB_lower']=bollinger['BBU_20_2.0']
    dataset['SMA_14']=dataset['SMA_14'].bfill()
    dataset['RSI_14']=dataset['RSI_14'].bfill()
    dataset['MACD']=dataset['MACD'].bfill()
    dataset['MACD-signal']=dataset['MACD-signal'].bfill()
    dataset['MACD-hist']=dataset['MACD-hist'].bfill()
    dataset['BB_upper']=dataset['BB_upper'].bfill()
    dataset['BB_lower']=dataset['BB_lower'].bfill()
    dataset.to_csv('BTCUSD_andicator.csv',index=False)
andicator()
```

شکل ۱۰ . کد پیاده سازی تابع andicator

۳.۱.۵ توابع seq و seq_prid :

معماری LSTM برای یادگیری وابستگی های طولانی مدت بین داده های قبلی و فعلی ساخته شده است مثلا اگر قیمت امروز به ۱۰ روز گذشته وابسته باشد این شبکه توانایی یادگیری این الگو را دارد بنابراین LSTM بر خلاف شبکه های معمولی نیازمند یک بعد دیگر در تنسور به نام sequence است نه تک نمونه سیکوئنس مشخص میکند شبکه به چند روز گذشته وابسته باشد تا الگو ها را یاد بگیرد. شبکه عصبی LSTM در پایتورچ ورودی هایی که دریافت میکند باید به شکل زیر باشد:

(batch_size,sequence_lenght,feauture_dim)

بنابراین با پیاده سازی این توابع شکل مناسب به تنسور داده ها، تا مناسب ورودی شبکه باشند. تابع seq برای تبدیل داده ها برای اموزش استفاده میشود که شامل داده های x و y است ولی در موقع پیش بینی ما فقط داده های xداریم و میخواهیم داده yپیش بینی کنیم پس نیازمند تبدیل داده y نیستیم. در تصویر زیر کد پیاده سازی این توابع آمده است:

```
def seq(x,y,seq_len):
    '''تبدیل داده به شکل مناسب برای ورودی lstm :
    (batch_size, seq_len, features)
    '''
    sequence=[]
    labels=[]
    for i in range(len(x)-seq_len):
        seq=x[i:i+seq_len]
        sequence.append(seq)
        label=y[i+seq_len-1]
        labels.append(label)
    return torch.stack(sequence),torch.tensor(labels,dtype=torch.float32)
def seq_prid(x,seq_len):
    '''تبدیل داده به شکل مناسب برای ورودی lstm :
    (batch_size, seq_len, features) برای پیش بینی'''
    if len(x)<seq_len:
        raise ValueError(f"data length {len(x)} is smaller than required seq_len {seq_len}")
    seq=x[-seq_len]
    return torch.tensor(seq, dtype=torch.float32).clone().detach().unsqueeze(0)
```

شکل ۱۱ . کد پیاده سازی توابع seq و seq_prid

۳.۱.۶ تابع PreData :

این تابع تابع اصلی پیش پردازش است که در آن ابتدا برای بررسی مقدار مینیمم و ماکزیمم هر ستون خروجی گرفته میشود که به شرح زیر است:

min and max of timestamp:1502928000000-1742515200000

min and max of open:3188.01-106143.8

min and max of high:3276.5-109588.0

min and max of low:2817.0-105321.49

min and max of close:3189.02-106143.82

min and max of volume:228.108068-760705.362783

min and max of close_time:1503014399999-1742601599999999

min and max of quote_asset_volume:977865.7333321-17465307097.88407

min and max of number_of_trades:2153-15223589

min and max of taker_buy_base:56.190141-374775.574085

min and max of taker_buy_quote:241363.80050245-8783916247.676138

min and max of ignore:0----0

min and max of SMA_14:3422.6978571428567-103578.42785714287

min and max of RSI_14:10.497797370369184-93.45927597449236

min and max of MACD:-4435.851793146449-6402.010446864906

min and max of MACD-signal:-4435.851793146449-6402.010446864906

min and max of MACD-hist:-4435.851793146449-6402.010446864906

min and max of BB_upper:3637.401000275189-108846.02334956186

min and max of BB_lower:3637.401000275189-108846.02334956186

با بررسی این اطلاعات مشخص میشود ستون ignore کلا برابر صفر است پس اطلاعات خاصی به ما نمیدهد پس میتوان حذف کرد همچنین در ستون close time ناهنجاری غیر متعارف و غ دیده میشود این ستون و ستون timestamp به شرط اینکه داده ها شافل نشوند و به صورت رندم وارد شبکه نشوند بلکه به صورت ترتیبی از اول وارد شوند تا ترتیب سری زمانی حفظ شود اگر داده ها شافل شوند مدل نمی تواند توالی داده هارا تشخیص دهد و باعث یادگیری اشتباه میشود.

در این تابع ابتدا داده ها به داده های وابسته (داده ای که قرار پیش بینی گردد یا همان y در این دیتا ست برابر ستون close است) و داده های غیر وابسته(داده ها و ویژگی های ورودی شبکه که مدل قرار است از این ویژگی یاد بگیرد و داده yرا پیش بینی کند، نام دیگر این داده ها همان x که در این دیتاست برابر تمامی ستون ها به جز timestamp, close (time, close, ignore تقسیم میشود در ضمن این داده ها شامل تمامی سطر های دیتاست اولیه نیست و از سطر ۱۲۵۰ به بعد هستند چون طبق تحقیقات بازار امروز یعنی ۲۰۲۵ بسیار متفاوت تر از ۲۰۱۷ است و الگو های بازار امروز بیشتر وابسته داده های جدید تر هستند. سپس داده ها به نسبت ۸۰-۱۰-۱۰ به ترتیب به مجموعه آموزش، تست، ارزیابی تقسیم شدند؛ در مرحله بعد داده ها توسط الگوریتم robust sacler مقیاس بندی و به تنسور تبدیل میشوند که در بخش ۲.۳ توضیحات کامل و نحوه عملکرد آن ارائه شده است پس از مقیاس بندی و فیت کردن با دیتاست از طریق فراخوانی توابع seq بعد توالی به تنسور ما اضافه میگردد و در مرحله اخر داده های ما چه اموزش چه تست و چه ارزیابی به مینی بچ ها تقسیم می شوند تا مدل پایدارتر و سریع تر یاد بگیرد. همچنین در اخر اسکالر های ما در یک فایل pkl ذخیره میشوند تا در موقع پیش بینی از پارامتر های استفاده گردد.

درتصویر زیر کد پیاده سازی این تابع آمده است:

```
def PreData(dataset):
    '''
        تقسیم مجموعه داده به تست و اموزش و ارزیابی
        مقیاس بندی داده
        ذخیره اسکالر
        تقسیم به بچ
    '''
    for colum in dataset.columns:
        print(f"min and max of {colum}:{dataset[colum].min()}----{dataset[colum].max()}")
    x=dataset.drop(columns=["timestamp","close_time","close","ignore"],inplace=False).iloc[1250:,:].values
    y=dataset.iloc[1250:,4].values
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,random_state=42,shuffle=False)
    sc_x=RobustScaler()
    sc_y=RobustScaler()
    x_train0=torch.tensor(sc_x.fit_transform(x_train),dtype=torch.float32)
    x_test0=torch.tensor(sc_x.transform(x_test),dtype=torch.float32)
    y_train0=torch.tensor(sc_y.fit_transform(y_train.reshape(-1,1)).flatten(),dtype=torch.float32)
    y_test0=torch.tensor(sc_y.transform(y_test.reshape(-1,1)).flatten(),dtype=torch.float32)
    print(f" x_train rang:{x_train0.min()}, {x_train0.max()}")
    print(f"y_train rang:{y_train0.min()}, {y_train0.max()}")
    print(f"x_test rang:{x_test0.min()}, {x_test0.max()}")
    print(f" y_test rang:{y_test0.min()}, {y_test0.max()}")
    x_trainsq,y_trainsq=seq(x_train0,y_train0,seq_len=110)
    x_testsq,y_testsq=seq(x_test0,y_test0,seq_len=110)
    x_trainsq0,x_valsq,y_trainsq0,y_valsq=train_test_split(x_trainsq,y_trainsq,test_size=1/9,random_state=42,shuffle=False)
    train_dataloeder=DataLoader(TensorDataset(x_trainsq0,y_trainsq0),batch_size=28,shuffle=False)
    val_dataloeder=DataLoader(TensorDataset(x_valsq,y_valsq),batch_size=28,shuffle=False)
    test_dataloeder=DataLoader(TensorDataset(x_testsq,y_testsq),batch_size=28,shuffle=False)
    try:
        joblib.dump(sc_x,"scaler_x.pkl")
        print("saved sclaer_x to: scaler_x.pkl")
        joblib.dump(sc_y,"scaler_y.pkl")
        print("saved scaler_y to : scaler_y.pkl")
    except Exception as e:
        print(f"Error saving scaler; {e}")
        raise
    return train_dataloeder,val_dataloeder,test_dataloeder
```

شکل۳.۱۲ . کد پیاده سازی تابع PreData

۳.۲ ماژول LSTM :

دومین مرحله مهم در یادگیری مدل آموزش مدل است قبل از اموزش مدل باید الگوریتم یادگیری ماشین یا شبکه عصبی مناسب را برای مسئله خود انتخاب و معماری آن را پیاده سازی کنیم سپس مدل را اموزش داده و حلقه اموزش و ارزیابی مدل و تست کردن عملکرد مدل میپردازیم، یکی از مهمترین مسائلی که در این بخش با ان مواجه میشود تنظیم هایپرپارامتر ها مثل تعداد لایه و نرخ یادگیری و... است افزایش پیچیدگی مدل باعث اورفتیتینگ مدل میشود که جلو مکانیزم هایی در این پروژه اعمال شده تا از این مشکل جلوگیری شود در عوض سادگی بیشتر مدل باعث اندرفیتیتگ میشود و مدل الگو هارا به خوبی یاد نمیگیرد پرامترهای مناسب با ازمون خطا تنظیم شده اند که جلوتر نتایج انها قرار داده شده و بررسی شده است.

۳.۲.۱ امپورت کتابخانه و توابع:

```
import joblib
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from DataPreprocesing import PreData,seq_prid
import pandas as pd
import numpy as np
```

شکل۳.۱۳ . امپورت کتابخانه ها و توابع مورد نیاز از ماژول DataPreprocesing

۳.۲.۲ کلاس شبکه LSTM :

این کلاس سنگ بنای مدل پروژه است، این کلاس از nn.Moudle ارث بری میکند که پایه تمام مدل های pytorchاست. این کلاس شامل تابع سازنده کلاس(constructor) است که ارگومان های ورودی آن شامل input size که تعداد ویژیگی هر تایم استنپ است، hidden layerکه شامل تعداد نورون ها در هر لایه LSTM است،output که شامل خروجی مدل مثل قیمت روز بعد و num layerکه شامل تعداد لایه های LSTM روی هم است.

لایه LSTM با با ورودی ۱۵ ویژگی و خروجی با ۵۰ نورون در هر گره و با تعداد لایه ۵ همچنین bach_first=True و با dropout برابر ۰.۴ تا از اورفیتینگ جلوگیری شئد تعریف شده است.

یک لایه fully connected که خروجی LSTMکه دارای ۵۰ نرون است به ۱۰۰ نرون تبدیل میکند لایه بعدی نرمال سازی بچ است تا سرعت یادگیری را افزایش دادع و از نوسان گرادیان جلوگیری کند. بین لایه اول و دوم fc از تابع فعال سازی Leaky ReLU استفاده شده است که در مورد آن در بخش ۲.۴ توضیحات کامل ارائه شده است هدف از استفاده این تابع جلوگیری از خروجی منفی است چون قیمت همواره مثبت است.

تابع بعدی که این کلاس شامل است تابع forwardاست که تعیین میکند داده ها چگونه از این لایه ها عبور کند؛ داده ها ابتدا از لایه LSTM عبور میکنند که خروجی آن شامل خروجی تمامی تایم استنپ ها در اخرین لایه LSTM است ولی در این

پروژه فقط خروجی اخرین تایم استنپ لازم است چون میخواهیم پیش بینی کنیم که بعد از دیدن کل توالی قیمت روز اینده چه خواهد بود سپس خروجی از لایه اول فولی کانکت عبور سپس خروجی ان وارد لایه نرمال سازی بچ میشود سپس تابع فعال سازی بر روی خروجی اعمال میشود و وارد لایه دوم میشود تا خروجی نهایی تولید گردد.

در تصویر زیر کد پیاده سازی این کلاس آمده است:

```python
class lstm_network(nn.Module):
    ''' کلاس شبکه عصبی lstm '''
    def __init__(self,input_size=15,hidden_layer=50,output=1,num_lay=5):
        super(lstm_network,self).__init__()
        self.LSTM=nn.LSTM(input_size,hidden_layer,num_lay,batch_first=True,dropout=0.4)
        self.fc1=nn.Linear(hidden_layer,100)
        self.bn=nn.BatchNorm1d(100)
        self.activation=nn.LeakyReLU()
        self.dropout=nn.Dropout(0.2)
        self.fc2=nn.Linear(100,output)
    def forward(self,x:torch.tensor)->torch.Tensor:
        out,_=self.LSTM(x)
        x=out[:,-1,:]
        p=self.fc1(x)
        p=self.bn(p)
        p=self.activation(p)
        p=self.dropout(p)
        pred=self.fc2(p)
        return pred
```

شکل ۱۴. کد پیاده سازی کلاس *LSTM*

۳.۲.۳تابع training :

در بخش قبلی کلاس شبکه پیاده سازی شد ولی این مدل بدون آموزش به درد نمیخورد در این تابع ابتدا مدل پیاده سازی شده به عنوان مدل آموزش تعیین میگردد سپس بهینه ساز که در این پروژه الگوریتم ADAM هست و همچنین این بهینه ساز و نحوه عملکرد آن در بخش ۲.۵ توضیح داده شده است و تابع هزینه که در این پروژه از تابع SmoothL1 استفاده شده تعیین میگردد و پس از آن بچ ها از طریق ماژول پیش پردازش وارد تابع میشوند سپس حلقه اموزش شروع میشود در هر اپوک با یک حلقه بچ ها به ترتیب وارد مدل میشوند و تابع هزینه محاسبه سپس بهینه ساز اعمال میشود تا وزن ها و بایاس اپدیت وگرادیان تعمیم گردد اگر تعداد حلقه کم باشد گرادیان تعمیم نمی یابد و مدل به خوبی الگو هارا یاد نمیگرد و اگر هم تعداد اپوک خیلی زیاد باشد ممکن است مدل به اورفیتینگ برسد؛ اورفیتینگ یعنی مدل انقدر به دیتاست فیت شود و الگو هارا یاد بگیرد که train loss به نزدیک صفر یا حتی صفر برسد این مدل اشتباه است چون موقع تست مدل تست لاست به شدت زیاد میشود چون مدل فقط الگو های دیتاست اموزش یاد گرفته و در دنیای واقعی به خوبی عمل نخواهد کرد و پیش بینی اشتباه خواهد داد، برای حل این مشکل در کلاس شبکه دراپ اوت تعیین گریدیده دراپ اوت به نسبت عددی که بهش داده میشود نورون های لایه رو غیرفعال میکند تا از این مشکل جلوگیری شود ولی در تعداد اپوک در این پروژه به صورت دستی مکانیزم early stop پیاده سازی شده است این مکانیزم به این صورت عمل میکند که بعد از هر بچ مدل با داده های ارزیابی،ارزیابی گردد و لاست آن ذخیره گردد اگر لاست جدید کمتر از بهترین لاست باشد لاست جدید ذخیره و جایگزین گردد سپس وزن و پارامترهای مدل در یک فایل pth ذخیره گردد تا دوباره در پیش بینی لود گردد همچنین یک patient تعریف گردیده که اگر مدل بیش از تعداد این متغییر لاست اپدیت نشود یعنی مدل به حد خود رسیده و حلقه متوقف گردد.پس از توقف حلقه ازمایش مدل دوباره از فایل بارگذاری گردیده و داده های تست روی آن ازمایش میگردد تا تست لاست بدست بیاید برای بررسی عملکرد مدل فقط تست لاست کافی نیست باید پارامتر های دیگر مانند MAPE,RMSE,SMAPE نیز بررسی گردد که نتایج را در بخش آزمون خطا بررسی خواهیم کرد.در تصویر زیر کد پیاده سازی این تابع امده است:

```python
def training():
    ''' آموزش مدل با داده هایی در ماژول DataPreprocesing
    بهینه ساز RMSprop شده
    تابع هزینه :SMOOTHL1LOSS استفاده شده
    با ارزیابی و تست شده
    '''
    start=time.time()
    torch.manual_seed(42)
    model=lstm_network()
    optimizer=torch.optim.Adam(model.parameters(),lr=0.001)
    loss_func=nn.SmoothL1Loss()
    train_dataloeder,val_dataloeder,test_dataloeder=PreData(pd.read_csv("BTCUSD_andicator.csv"))
    val_loss_values=[]
    loss_val=[]
    best_val_loss=float('inf')
    patient=10
    counter=0
    epochs=[]


    for epoch in range(300):
        model.train()
        train_loss_epoch=0.0
        for batch_idx,(inputs,label) in enumerate(train_dataloeder):
            inputs,label=inputs.float() , label.float().unsqueeze(1)
            optimizer.zero_grad()
            outputs=model(inputs)
            loss=loss_func(outputs,label)
            loss.backward()
            optimizer.step()
            train_loss_epoch+=loss.item()
        train_loss_epoch/=len(train_dataloeder)
        loss_val.append(train_loss_epoch)

        model.eval()
        val_loss=0
        with torch.no_grad():
            for inputs,label in val_dataloeder:
                outputs=model(inputs)
                loss=loss_func(outputs,label.unsqueeze(1))
                val_loss+=loss.item()
        val_loss/=len(val_dataloeder)
        val_loss_values.append(val_loss)
        epochs.append(epoch)
        print(f" epoch {epoch+1}/1000 , train loss: {train_loss_epoch:.4f} , val loss; {val_loss}")
        if val_loss < best_val_loss and epoch>=5:
            best_val_loss=val_loss
            counter=0
            torch.save(model.state_dict(),"lstm.pth")
            print(f"model saved with val loss: {best_val_loss}")
        else:
            counter+=1
            if counter >= patient:
                print(f"early stopping triggered after {epoch+1} epochs")
                break
    model.load_state_dict(torch.load("lstm.pth",weights_only=True))
    model.eval()
    test_loss_total=0
    mape_total=0
    n_sample=0
    rmse_total=0
    smape_total=0
    with torch.no_grad():
        for inputs,label in test_dataloeder:
            y_pred=model(inputs)
            test_loss=loss_func(y_pred,label.unsqueeze(1))
            test_loss_total+=test_loss.item()
            eps=1e-8
            mape=torch.abs((y_pred-label.unsqueeze(1)) / (label.unsqueeze(1) + eps))
            mape_total+=mape.sum().item()
            rmse=((y_pred - label.unsqueeze(1))**2).sum().item()
            rmse_total+=rmse
            smape = torch.abs(y_pred - label.unsqueeze(1)) / ((torch.abs(y_pred) + torch.abs(label.unsqueeze(1))) / 2 + ep
            smape_total += smape.sum().item()
            n_sample+=label.size(0)
    avg_test_loss=test_loss_total/len(test_dataloeder)
    print(f'Test Loss: {avg_test_loss:.4f}')
    print(f'MAPE: {(mape_total / n_sample) * 100:.2f}%')
    rmse = math.sqrt(rmse_total / n_sample)
    print(f'RMSE: {rmse:.4f}')
    smape_avg = (smape_total / n_sample) * 100
    print(f'SMAPE: {smape_avg:.2f}%')

training()
```

شکل ۱۵. کد پیاده سازی تابع *training*

۳.۲.۴ آزمون خطا مدل:

برای رسیدن مدل به بهترین حالت و پایدارترین حالت خود نیازمند آزمون و خطا و بررسی عملکرد مدل است؛ تمامی پرامترهایی که در پیاده سازی های بالا تعیین گردیده است به صورت رندم انتخاب نشدن با ازمون خطا تنظیم شده اند تا مدل به کمترین تست لاست و بهترین عملکرد خود برسد در عین حال دچار اورفتیتنگ و اندرفیتینگ نگردد. ابتدا پارامتر ها به صورت رندم و کم تنظیم شدند سپس تا ازمون خطا مقدارشون کاهش یا افزایش پیدا کرده که همراه با mape به شرح زیر است:

Hidden layer=10 / num layer=2/ liniear=100/

Patient=10/dropout=0.2-> mape=61%

Hidden layer=20->mape=75%

Hidden layer=30->mape=66%

Hidden layer=40->mape=127%

Hidden layer=50->mape=38.8%

Num layer=3->mape=59%

Num layer=4->mape=43%

Num layer=5->mape=19%

Dropout=0.3->mape=29%

Drop out=0.4->mape=4%

با بررسی های بالا مشخص گردید که در اخرین آزمون mapeبرابر ۴ درصد شده که یعنی مدل به بهترین عملکرد خود رسیده است به خصوص در مسائلی همچون سری زمانی که دارای نوسان خیلی زیاد هستند به خصوص قیمت رمز ارز ها که سال اخیر نوسانات خیلی شدید تری تجربه کرده در کل mapeزیر ۵ درصد برای مدل های سری زمانی به خصوص بیت کوین ایده آل است و میتونه پیش بینی نسبتا دقیقی داشته باشه و روند بازار درک بکند. در این بهترین مدل سایر پارامتر های عملکرد به شرح زیر است:

Test Loss: 0.0147

MAPE: 8.10%

RMSE: 0.1870

SMAPE: 8.44%

تست لاست مشخص میکند فاصله مطلق داده پیش بینی شده با داده واقعی چقدر است که در اینجا برابر ۰.۰۱۴۷ است عدد قابل قبول و حتی خوبی برای داده های پرنوسان است.

Mape میانگین درصد خطای مطلق را نشان میدهد یعنی به طور متوسط پیش بینی ها ۸.۱ درصد با مقادیر واقعی اختلاف دارند

Rmse وزن بیشتری به خطاهای بزرگ میدهد و مشخص میکند خطاهای بزرگ چقد روی متوسط کل تاثیر گذاشته که در اینجا برابر۰.۱۸۷ است تقریبا ۱۸ درصد از کل دامنه

Smape نسبت به mape در برخورد با مقادیر نزدیک صفر و نوسانات شدید متقارن تر است و عدد ۸.۴۴ درصد نشان میدهد مدل تعادل مناسبی بین پیش بینی های بیش آورد و کم آورد حفظ کرده است.

۳.۲.۵ تابع predict:

در این تابع از مدل اموزش دیده استفاده میکنیم تا با داده های جدید پیش بینی های جدید انجام بدهیم؛ ابتدا مدل بارگذاری میگردد سپس دیتاهایی که میخواهیم پیش بینی روی آنها انجام گردد وارد میشوند باید به تعداد سیکوئنس باشند که مدل با آن آموزش دیده چون مدل با توالی سیکوئنس الگو هارا یاد گرفته است سپس پیش پردازش هایی که برای اموزش مدل استفاده شده روی داده های جدید اعمال میکنیم پس از اماده سازی داده ها آنها وارد شبکه عصبی و جریان آ«ها به صورت تابع forwardخواهد بود و پس از آن خروجی آن به صورت خام تحویل میدهد برای تبدیل به قیمت های واقعی باید از اسکالر های ذخیره شده استفاده گردد تا خروجی به قیمت اولیه یعنی قبل از مقیاس بندی برگردد در صفحه بعدی تصویر کد پیاده سازی این تابع آمده است:

```python
def predict():
    '''پیش بینی قیمت یک روز بعد با استفاده از داده های 110 روز قبل'''
    model=lstm_network()
    model.load_state_dict(torch.load('lstm.pth',weights_only=True))
    model.eval()
    data=pd.read_csv("BTCUSD_andicator.csv").tail(110)
    print(data.tail(5)[["timestamp","close"]])
    print(f"min close: {data['close'].min()}")
    print(f"max close: {data['close'].max()}")
    print("input size:", data.drop(columns=["timestamp","close_time","close","ignore"],inplace=False).shape[1])
    sc_y=joblib.load("scaler_y.pkl")
    sc_x=joblib.load("scaler_x.pkl")


    data1=sc_x.transform(data.drop(columns=["timestamp","close_time","close","ignore"],inplace=False).values)
    print(F'min/max data:\n min: {data1.min()}\nmax: {data1.max()}')
    data2=seq_prid(data1,seq_len=110).unsqueeze(0)
    with torch.no_grad():
        prediction=model(data2)
    print(f"Raw output: {prediction}")
    org_pred=sc_y.inverse_transform(prediction.detach().numpy())
    return org_pred[-1].item()
print(predict())
```

شکل۱۶ . کد پیاده سازی تابع predict

اخرین داده موجود در دیتاست برابر تاریخ 2025/03/21 است که در آن قیمت بیت کوین در کلوز برابر 84088 است طبق مستندات بایننس قیمت کلوز بیت کوین در تاریخ 2025/03/22 برابر 84584 است قیمتی که این تابع در خروجی میدهد برابر87736.7890625 که با mape=8% انطباق دارد ولی چون قیمت بیت در حول نزدیک ۱۰۰ هزار است حتی اگر خطا یک درصد باشد برابر هزار دلار خواهد بود؛قیمت در تاریخ۲۲ جهش داشته و مدل ما افزایش قیمت پیش بینی کرده یعنی الگو روند بازار را مدل به خوبی یاد گرفته است.

۳.۳فاین تیونینگ:

در بخش قبل دیدیم که مدل قیمت روز اینده یعنی 2025/03/22 را پیش بینی کرد ایا میتوان برای پیش بینی 23ام یا 30ام از مدل استفاده کرد ؟مدل ها به خصوص مدل های پیش بینی سری زمانی بیت کوین برای پیش بینی جدید نیازمند اپدیت دیتاست و اپدیت وزن و بایاس و گرادیان خود هستند تا الگو داده های جدید را نیز یاد بگیرند تا بتوانند توالی ایجاد کرده و پیش بینی انجام بدهند. فاین تیونینگ مرحله ای از یادگیری عمیق است که در آن یک مدل از پیش آموزش دیده را با داده های خاص یا داده های تازه دوباره آموزش میدهیم تا عملکرد مدل در آن دامنه بهینه گردد.به جای آموزش مدل از صفر که نیازمند حجم زاد داده و مصرف منابع دارد با یک مدل از پیش آموزش دیده شروع کرده و تنها آنرا روی داده های جدید خود بروزرسانی میکنیم. در این ماژول توابع ,LastUpdate_read,SaveUpdate,NewData UpdateDataset,Fine_tune پیاده سازی شده است.

۳.۳.۱ امپورت های این ماژول:

```python
import os
import requests
import pandas as pd
import torch
import torch.nn as nn
from lstm import lstm_network,predict
from DataPreprocesing import PreData,andicator,seq_prid
import math
```

۳.۳.۲ توابع lastupdate,saveupdate:

در تابع LastUpdate_read تاریخ اخرین بروزرسانی اپدیت دیتاست که در یک فایل تکست ذخیره شده را بازیابی میکند.

تابع SaveUpdate نیز تاریخ اخرین اپدیت جدید را در داخل فایل تکست ذخیره میکند.

```python
def LastUpdate_read():
    """خواندن تاریخ اخرین اپدیت"""
    try:
        with open("LastUpdate.txt","r") as file:
            timestamp=int(file.read().strip())
        return timestamp
    except (FileNotFoundError,ValueError):
        return print("file not found,value error")

def SaveUpdate(timestamp):
    '''ذخیره تاریخ اپدیت به عنوان اخرین اپدیت'''
    with open("LastUpdate.txt","w") as file:
        file.write(str(int(timestamp)))
```

شکل۱۸ . کد پیاده سازی توابع LastUpate_read,SaveUpdate

۳.۳.۳ توابع Newdata,Updatedataset

در این توابع با استفاده از تاریخ اخرین اپدیت، داده های جدید از طریق API بایننس دریافت و با دیتاست قدیمی الحاق و یک دیتاست جدید و اپدیت ایجاد میکنند و اخرین تایم استنمپ موجود در دیتاست به وسیله تابع SaveUpdate ذخیره میکنند کد پیاده سازی آن به شکل زیر است:

```python
def NewData(lastupdate):
    '''دریافت داده های بروز از بایننس'''
    proxy={

    }
    urls=f"https://api.binance.com/api/v1/klines?symbol=BTCUSDT&interval=1h&startTime={lastupdate}"
    response=requests.get(url=urls,proxies=proxy,timeout=10)
    if response.status_code==200:
        print("The requests was successful")
    else:
        print(f"request faild! status code: {response.status_code}")
    data=response.json()
    if not data:
        print("No data recevied from api")
        return None
    df=pd.DataFrame(data,columns=['timestamp', 'open', 'high', 'low', 'close', 'volume', 'close_time',
                                  'quote_asset_volume', 'taker_buy_base', 'taker_buy_quote', 'number_of_trades'
        , 'ignore'
    ])
    return df
def UpdateDataset():
    '''افزودن داده های جدید به دیتاست قدیمی'''
    lastupdate=LastUpdate_read()
    new_data=NewData(lastupdate)
    if new_data is not None and not new_data.empty:
        dataset=pd.read_csv("BTCUSD_combine.csv")
        dataset=pd.concat([dataset,new_data],ignore_index=False).drop_duplicates(subset=["timestamp"])
        dataset.to_csv("BTCUSD_update.csv",index=False)
        SaveUpdate(new_data["timestamp"].max())
```

شکل۱۹ . کد پیاده سازی توابع NewData,UpdateDataset

۳.۳.۴ تابع Fine_tune :

همانند تابع trainingاست ابتدا مدل بارگذاری و بهنیه ساز و تابع هزینه اعمال میشود سپس دیتاست جدید با همان پیش پردازشی که برای آموزش انجام دادیم پیش پردازش میشود سپس حلقه اپوک ایجاد میشود ولی چون فقط نیازمند اپدیت مدل هستیم تعداد اپوک و پیشنت را بسیار کمتر تعیین میکنیم همچنین مقدار lr در بهینه ساز را عدد کوچیکی قرار میدهیم تا مدل به صورت نرم اپدیت شود و تعمیم گردد.

ل

با تشکر

۴. کلام اخر:

بازار های مالی به خصوص رمز ارز ها مانند بیت کوین بسیارپرنوسان هستند و عوامل مختلفی بر روی آنها تاثیر گذار است یکی از این عوامل احساسات است مثلا توییت و یا اعلام تحریم از سوی یک کشور موجب ایجاد نوسان در بازار میگردد و یا افزایش نرخ بیکاری یا ریسک پذیری مردم موجب تغییر قیمت در بازار های مالی میشود بنابراین مدل های مبتنی بر یادگیری عمیق از داده های عددی استفاده میکنند تا روند بازار و یا قیمت را پیش بینی کنند ولی توانایی پیش بینی موارد بالا را ندارند بنابراین در این مدل ها عملکرد مدل در دنیای واقعی ممکن است درست نباشد ولی با اپدیت مدل با فاین تیونینگ و یا استفاده از مدل های هیبریدی و یا یادگیری انتقالی میتوان عملکرد و نتیجه مطلوب تری داشت، در این پروژه سعی بر این شد تا با استفاده از دیپ لرنینگ و شبکه عصبی LSTM روند بازار و قیمت اینده تا حدودی پیش بینی شود.