Zelta Labs Untrade Crypto Trading Challenge

Team 46

Table of Contents

1. Introduction

This report aims to elucidate our trading strategy for the BTC/USDT cryptocurrency market by exploring the development process, risk management techniques, backtesting analysis, and the overall insights gained from this project.

Following much deliberation and experimentation, we decided to model it as a classification problem to predict the direction of BTC's price, rather than the actual price itself.

Our final strategy consists of an ensemble of classifiers assisted with a few technical indicators, which capitalise on both long and short trades.

2. Rationale behind Classification

Our initial attempts to forecast BTC's price did not yield promising results. Even strategies solely reliant on technical analysis lacked reliability.

Price prediction models for BTC face challenges due to high volatility, absence of clear fundamental factors and the influence of market sentiment and speculation. These factors make accurate and consistent predictions difficult.

While still employing prediction techniques may be possible, we encountered challenges in establishing significant relationships and preventing overfitting.

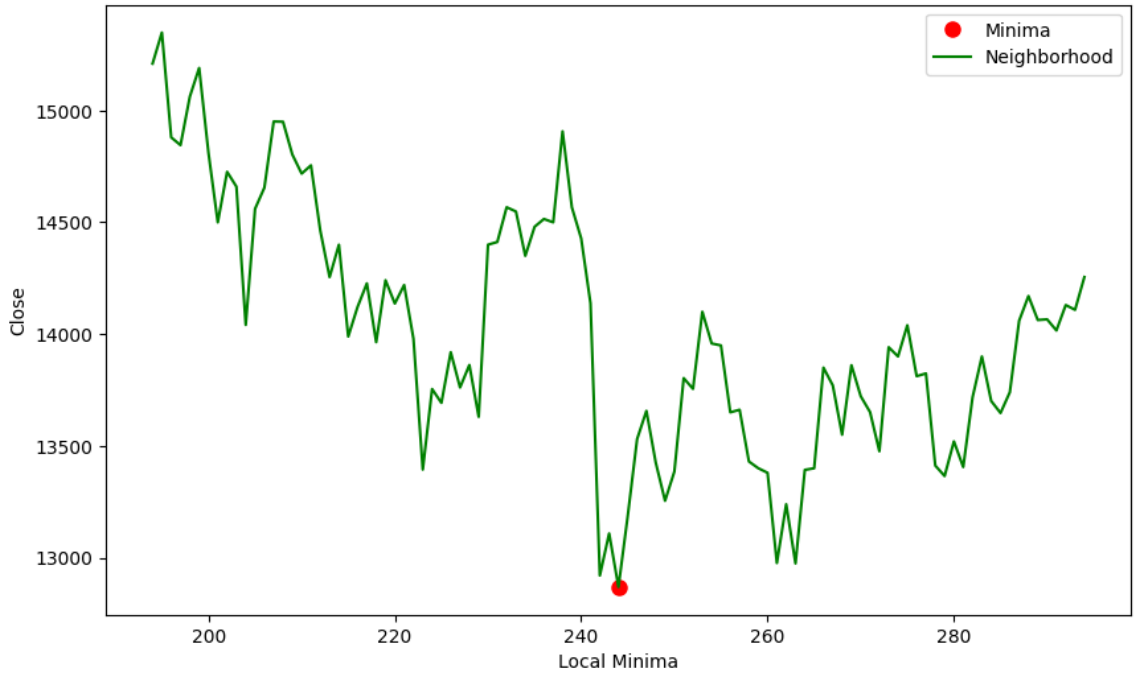
3. Trading Conditions

1. OHLCV data time frame - **1 hour**
2. Types of trades - **both long and short**
3. Buying/selling at **close price**

4. Strategy Logic

The strategy attempts to classify extremas and use them as entry and exit points for trades.
In this context, an extrema is defined in a neighbourhood of a specified size.

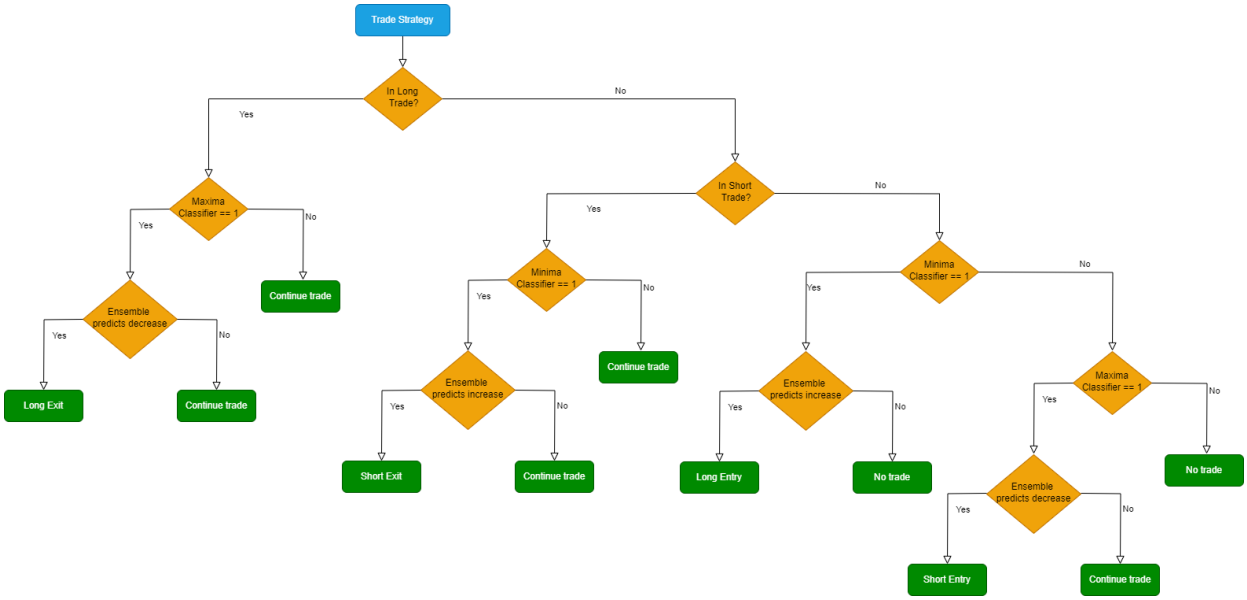
Let us consider a window size of 24. Then a minima would be defined as a point that is smaller than 24 points to its left and 24 points to its right. A maxima is defined likewise.



The strategy works as follows:

1. **Train Extrema Classifiers** - We build two separate classifiers - one for minima and one for maxima, using random forests.

2. **Train Neural Network for Single Timestamp Classification** - We build a binary classifier to predict the increase or decrease in price for a single timestamp.
3. **Train LSTM Model for Single Timestamp Classification** - We build another binary classifier for single timestamp direction.
4. **Defining Entry and Exit Conditions -** For entry into a long trade, if we identify a minima using the extrema classifier, we then use an ensemble of the two single step classifiers to verify an upcoming increase in price, and take a long position. Similarly, for exiting a long trade, we identify a maxima and assist it using the single step ensemble.
   The conditions are reversed for short trades.
5. **Stop Loss -** In case that our predictions were wrong, and prices start falling (or rising for short positions) beyond a certain threshold, we exit the trade to contain our losses.



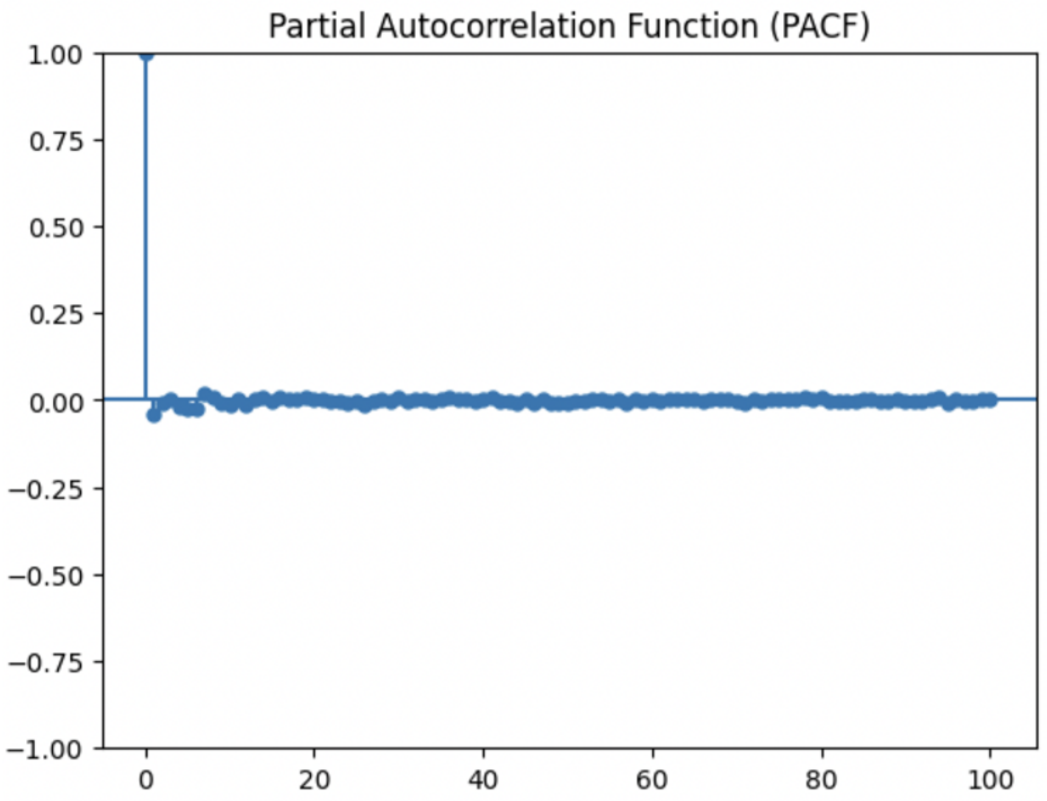
We'll now take a look at each step in detail.

5. Neural Network Single Step Classifier

5.1 Properties Exploited

1. **Stationarity of First Order Difference:** The closing price time series is not stationary which is concluded using an Augmented Dickey Fuller test.
   However, the test shows that the first order difference of closing price time series is stationary.
   Hence, the joint distribution of the subset of time series of first order difference $X_i, X_{i+1}, \ldots, X_{i+t}$ is approximately the same for all i when a t is fixed.

2. Values in the random process of first order difference of time series are asymptotically independent. The PACF plot for 1 hour sampling period data is decaying.



5.2 Modelling Problem as Classification

5.2.1 Target - Use a model to infer whether the price will increase or decrease in the next time step.

We need the probability of the next timestep's first-order difference being greater or less than equal to zero. $X(t)$ represents the first-order difference of the closing price time series. The classifier model approximates the following probability:

$$P(X(t) > 0 | X[t-d : t-1])$$

This can be done because of the two properties mentioned.

5.2.2 Features - The features taken were the first-order differences of past 'd' values. The value of d (number of lags taken as feature) is taken as an empirical value.

5.2.3 Classifiers Tried - Random Forest, Logistic Regression, SVM, LSTM and a Neural Network were experimented with.

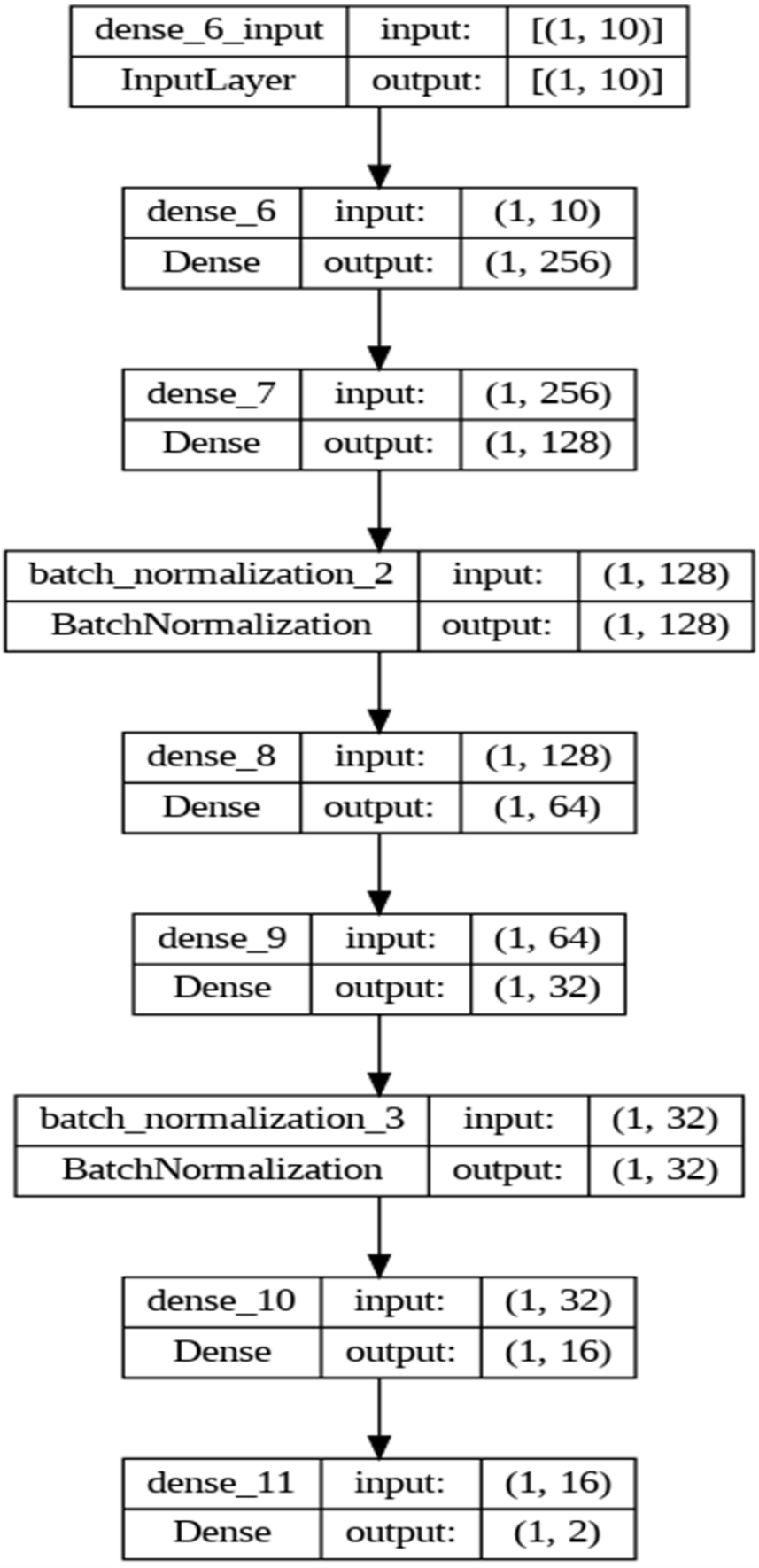
5.2.4 Data Preparation - A set of $X[t - d : t - 1]$ labels as +1 (price increase) and -1 (price decrease).
Train and test data were formed by randomly sampling from this set with 0.33 probability being for the test set.

5.2.5 Results and Experiments - The value of 'd' chosen was 10. Therefore, the past 10 first-order differences are taken as features.

Random forest exhibited overfitting on the train set with an accuracy of around 51% on the test set.
LSTM failed to converge on the train set after tuning for hyperparameters too.

Out of the experiments performed, we obtained the best results of 55% accuracy using a Neural Network which was deep enough to learn from the train set and also not overfit on test data. Batch Norm was used for improvement.

Block Diagram of the Neural Network Classifier Model


dense_6_input
input:
[(1, 10)]
InputLayer
output:
[(1, 10)]
dense_6
input:
(1, 10)
Dense
output:
(1, 256)
dense_7
input:
(1, 256)
Dense
output:
(1, 128)
batch_normalization_2
input:
(1, 128)
BatchNormalization
output:
(1, 128)
dense_8
input:
(1, 128)
Dense
output:
(1, 64)
dense_9
input:
(1, 64)
Dense
output:
(1, 32)
batch_normalization_3
input:
(1, 32)
BatchNormalization
output:
(1, 32)
dense_10
input:
(1, 32)
Dense
output:
(1, 16)
dense_11
input:
(1, 16)
Dense
output:
(1, 2)

6. LSTM Single Step Classifier

6.1 Reason for Choosing LSTM Model

The LSTM (Long Short-Term Memory) model was selected primarily due to its capability to handle time series data efficiently. It excels in capturing long-range dependencies and patterns within sequential data like stock prices. Its ability to retain and selectively process information over extended time intervals makes it suitable for predicting BTC price movements over 1-hour intervals.

6.2 Classification Problem Description

6.2.1 Problem Statement

The classification problem involves predicting whether the BTC price will increase (Class 1) or decrease (Class 0) based on 1-hour BTC data.

6.2.1 Features Used

- Previous 60 hours of BTC price and volume
- ETH price and volume

6.3 Model

6.3.1 Model Architecture Description

The LSTM (Long Short-Term Memory) model comprises multiple layers for sequential data processing.
The input layer is configured with dimensions of (60, 4) for the features.

Layer 1: LSTM (128 neurons)

- This layer utilises 128 LSTM units to process the input sequences.

- Dropout: 0.2 - This implies a 20% dropout rate, randomly dropping 20% of units during training to reduce overfitting.
- Activation Function: tanh
- Batch Normalisation: Applied after the Dropout layer to normalize the inputs.

Layer 2: LSTM (128 neurons)

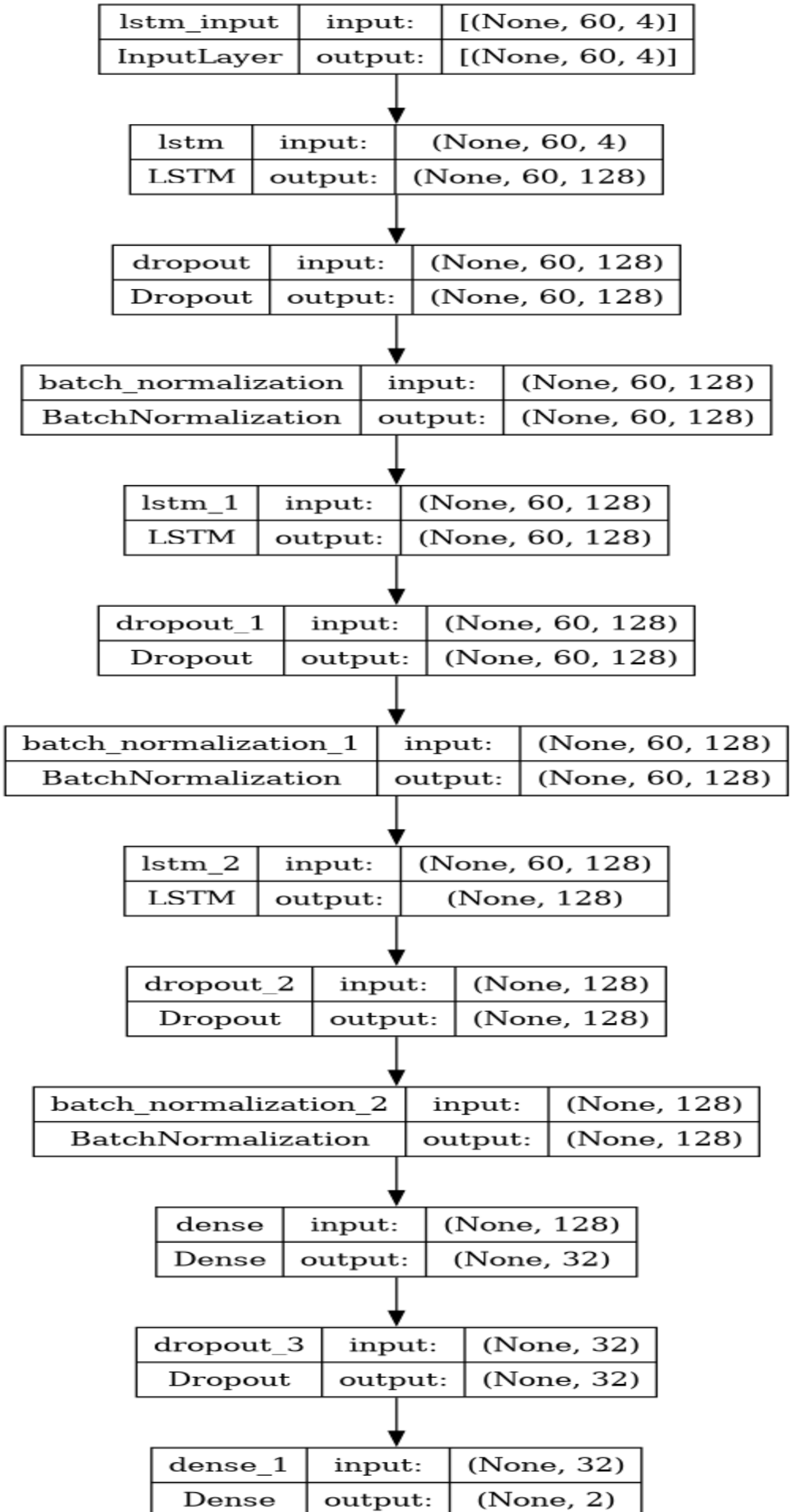
- Similar to the previous LSTM layer, it consists of 128 LSTM units.
- Dropout: 0.1 - A lower dropout rate of 10% to further prevent overfitting.
- Activation Function: tanh
- Batch Normalisation: Employed after the Dropout layer for normalization.

Layer 3: LSTM (128 neurons)

- Comprising 128 LSTM units.
- Dropout: 0.2 - A dropout rate of 20% applied to mitigate overfitting.
- Activation Function: tanh
- Batch Normalisation: Applied after the Dropout layer for data normalization.

Additional Layers

- Dense Layer (32 neurons, ReLU activation): A densely connected layer with 32 neurons and Rectified Linear Unit (ReLU) activation function.
- Dropout: 0.2 - 20% dropout to prevent overfitting in this densely connected layer.
- Output Layer (2 neurons, softmax activation): Outputs a probability distribution over the two classes (price increase or decrease).


lstm_input | input: | [(None, 60, 4)]
InputLayer | output: | [(None, 60, 4)]
lstm | input: | (None, 60, 4)
LSTM | output: | (None, 60, 128)
dropout | input: | (None, 60, 128)
Dropout | output: | (None, 60, 128)
batch_normalization | input: | (None, 60, 128)
BatchNormalization | output: | (None, 60, 128)
lstm_1 | input: | (None, 60, 128)
LSTM | output: | (None, 60, 128)
dropout_1 | input: | (None, 60, 128)
Dropout | output: | (None, 60, 128)
batch_normalization_1 | input: | (None, 60, 128)
BatchNormalization | output: | (None, 60, 128)
lstm_2 | input: | (None, 60, 128)
LSTM | output: | (None, 128)
dropout_2 | input: | (None, 128)
Dropout | output: | (None, 128)
batch_normalization_2 | input: | (None, 128)
BatchNormalization | output: | (None, 128)
dense | input: | (None, 128)
Dense | output: | (None, 32)
dropout_3 | input: | (None, 32)
Dropout | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 2)

6.3.2 Dropout and Batch Normalisation

- Dropout layers (0.2, 0.1, 0.2) were added to mitigate overfitting by randomly dropping units during training.
- Batch Normalisation layers were introduced to normalize and stabilize learning in each layer of the network.

6.3.3 Learning Rate Schedule

An Exponential Decay learning rate schedule was used with an Adam optimizer to aid in convergence and prevent overfitting.

- Initial learning rate: Initialised to 0.001, this represents the initial learning rate at the start of training.
- Decay steps: Set to 10,000, defining the intervals at which the learning rate decays.
- Decay rate: Fixed at 0.9, indicating the rate at which the learning rate diminishes, decreasing by 10% at every decay_steps.

6.3.4 Scaling Features

The percentage change in features was computed and scaled appropriately to facilitate model training and testing.

6.4 Training and Results

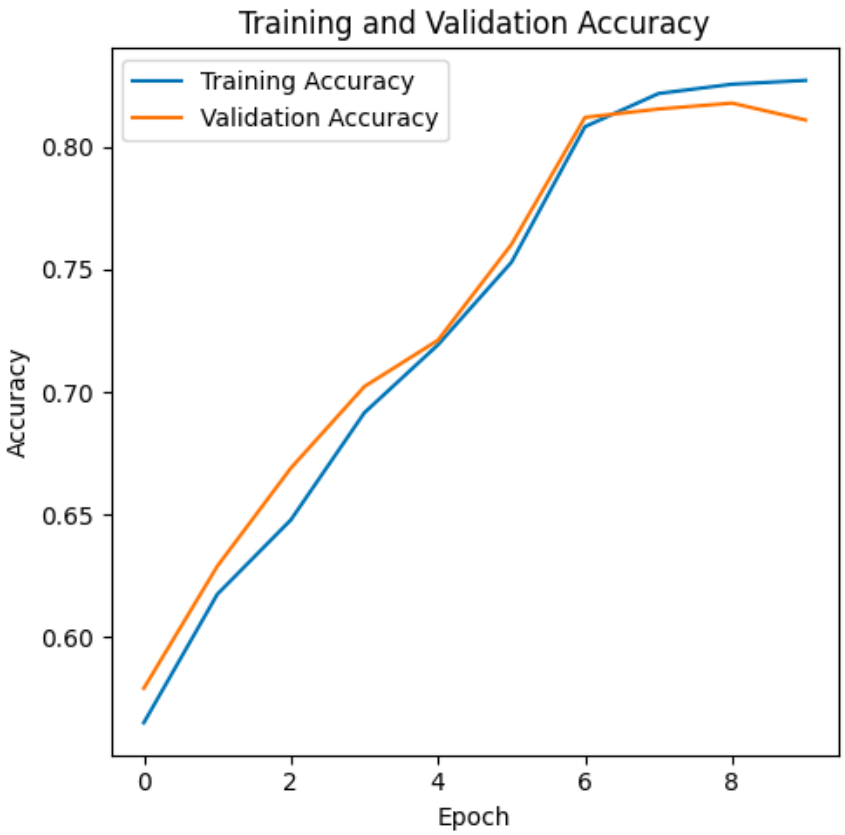
6.4.1 Training Procedure

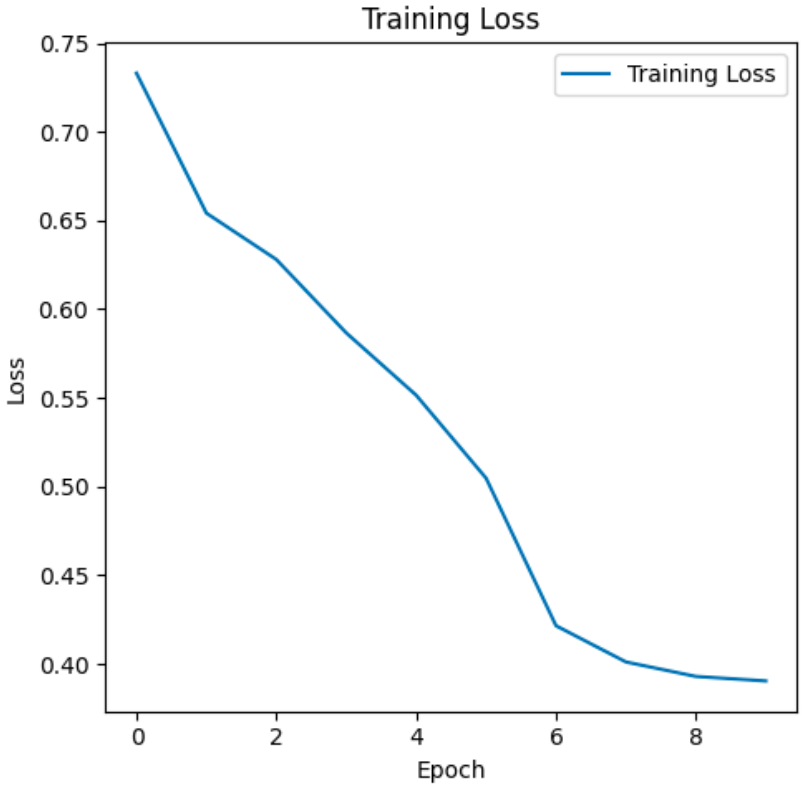
The training process involves splitting the dataset into training and validation sets using a 70-30% ratio.

6.4.2 Results and Performance Metrics

- The model achieved an accuracy of 82.72% on the training data and 82.63% on the validation set after 10 epochs.
- Overfitting was managed through the application of dropout layers, resulting in better generalization.

- The validation accuracy slightly surpassed the training accuracy due to the dropout layers, indicating the model's improved robustness during testing.


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
Epoch


Training Loss
Training Loss
Loss
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
2
4
6
8
Epoch

7. Extrema Classifier

Two separate models were trained here - one for minimas and another for maximas.

7.1 Target

For a fixed window size ‘w’, use a model to infer whether the price will be the minimum/maximum for ‘w’ timestamps before and after.

7.2 Features

Price momentum is calculated as the moving average of percent change in price for the last ‘d’ timestamps.
Aside from price momentum, RSI, Stochastic Oscillator and MACD indicators were used as features. Let $X(t)$ be the set of these features at time t.

7.3 Data Preparation

A set of $X[t - d : t - 1]$ with labels as 1 (extremum) and 0 (non-extremum).
Train and test data were formed by randomly sampling from this set with 0.3 probability being for the test set.

7.4 Result and Experiments

For w = 5, d = 10, we get an accuracy of 93%. However accuracy is not a very useful feature here as we would have a lot more 0s than 1s. Sensitivity (TP / TP + FN) is a better metric here.
We obtained a result of 70% sensitivity on the given data.

8. Risk Management

8.1 During Entry

Entry conditions were made very strict. For example, for a long entry, if a minima is identified by the classifier, it must be assisted by the ensemble of single step classifiers in order to enter the trade. If the neural network classifier predicts an increase, and the LSTM classifier predicts an increase with a confidence level of 80%, then we enter the trade.

The models were trained independently on different features. Their ensemble can increase the win rate probability significantly.

8.2 Containing Drawdown

A stop-loss of 1% was used to contain our losses.
Hence, the model would exceed 10% drawdown if we had around 10 consecutive losses.
Our model had a win rate of 84% during backtesting for 393 trades.
Even if we assume a lower win rate of 70% for 500 trades, the probability of getting 10 consecutive bad trades is only around 0.2%.

Hence, this simple stop-loss strategy should be able to contain overall drawdowns, as seen in the backtesting where we achieved 5.69% drawdowns.

9. Overfitting

We tried the following methods to avoid overfitting:

- Added dropout layers to the LSTM model
- The neurons can't rely on one input because it might be dropped out at random. This reduces bias due to over-relying on one input, bias is a major cause of overfitting.
- Neurons will not learn redundant details of inputs. This ensures only important information is stored by the neurons. This enables the neural network to gain useful knowledge which it uses to make predictions.
- The ensemble of classifiers used can reduce overfitting as each classifier is trained on different features. This diversity in learning helps reduce overfitting because individual models may overfit to different subsets or aspects of the data.
- For random forests in particular, pruning can be used to limit the depth of each decision tree.

10. Backtesting Results

1. **Gross Profit** - 19368.55437236073
2. **Net Profit** - 18754.22706466467
3. **Total Closed Trades** - 393
4. **Win Rate (Profitability %)** - 84.478372
5. **Max Drawdown** - (-5.6988902862969044 %)
6. **Average Winning Trade (in USDT)** - 83.48476761948934
7. **Average Losing Trade (in USDT)** - (-17.790388433018934)
8. **Buy and Hold Return of BTC** - 215.43328 %
9. **Largest Losing Trade (in USDT)** - (-58.98890286296912)
10. **Largest Winning Trade (in USDT)** - 493.1133542942246

11. Insights Gained

1. A strategy that has a combination of long and short trades is more likely to perform better than one that only has one type, as it can perform well in both bullish and bearish periods.

2. Commonly used approaches in the stock market need not work well in the cryptocurrency market. Technical analysis methods alone don't give great results, but they can be useful if used in conjunction with other approaches.

12. Alternate Approaches

Time series prediction models like ARIMA, VAR and LSTM were tried in a next time step prediction setup. These models were not very effective as reflected by the RMSE values.
ARIMA and VAR are linear models based on the stationarity assumption. The main reason behind the ineffectiveness of ARIMA and VAR was the absence of any significant value in any of the lags in PACF and ACF plots.
Also, when we apply the box-cox transform to make the distribution of values closer to the Gaussian distribution, it doesn't change the scenario.

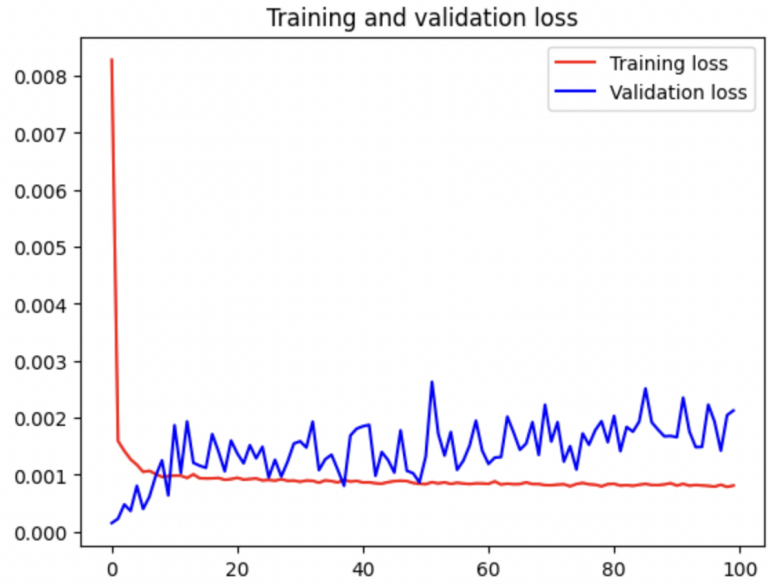
Results with configuration details-

1. ARIMA with a configuration of (5,1,4) for (autoregressive lag, integral, moving average lag) was applied to the BTC time series data with a 5-minute sampling period. An RMSE value

of 106.57 is obtained on the 20% test set which is from the end of the 18-22 data. If we rely on the prediction of price for the next time step and decide whether the price will be decreasing or increasing, we get an accuracy of around 49-50%.

2. The VAR model is also experimented upon. The multiple time series are first-order differences of - closing price, opening price, highest price, lowest price and volume. The prediction of the closing price is studied. The RMSE value for the test data obtained by taking 49 lags is around 200.

3. LSTM was tried with a combination of a different number of layers and a dropout. But it overfitted on the training set giving a high RMSE of around 2000 on the test set. The testing set is the last 30% of the 2018 - 2022 time series.



Loss curve for LSTM

We also tried out technical analysis-based strategies such as RSI divergence, mean reversion, and other momentum approaches, however none of those could give satisfactory results without overfitting.