Using Bitcoin Pricing Data to Create a Profitable Algorithmic Trading Strategy

By Motati Dinesh Reddy

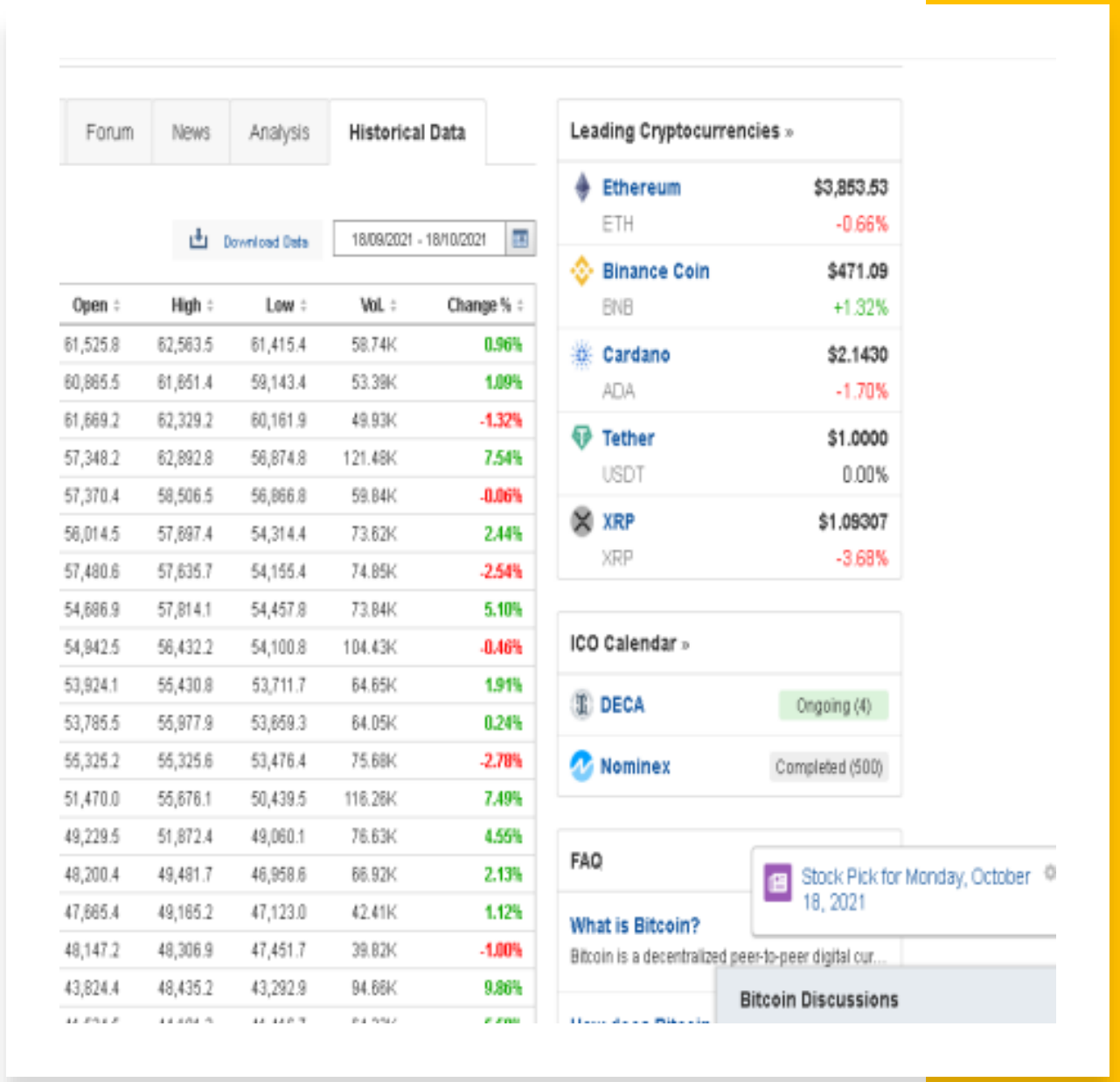
INTRODUCTION

- Bitcoin and other cryptocurrencies have high volatility in their pricing, meaning that prices can fluctuate significantly over a short period.
- In this project, we created an algorithm that would predict the price of bitcoin in x minutes relative to the current price


STEPS
STEP 1 DATA COLLECTION
STEP 2 DATA CLEANING AND EDA
STEP 3 FEATURE SELECTION AND TRAINING MODEL
STEP 4 MODEL TESTING AND VALIDATION
STEP 5 MODEL PREDICTION

- DATA SET

DATA IS TAKEN FROM BITCOIN HISTORICAL DATA Contains Timestamp, open , high , low , volume(currency), weighted price, change% from 1-07-21 to 18- 10-21



- Features

- We decided to make a total of 66 features for each time step based on the last n minutes, where n =1,2,5,10,20,40,80.
- H: (High price over last n minutes)/Current price
- • L: (Low Price over last n minutes)/Current price
- • A: (Average Price over last n minutes)/Current price
- • V: Volume of trading in last n minutes in BTC
- • P: Proportion of increases in price every minute over last n minutes
- •AC R: Ratio of price n minutes ago to the current price
- • WAP: (Average price over [t-2n, t-n])/(Average price over [t-n, t])
- • AP: (Average price over [t-2n, t-n])/Current price
- • VR: (Volume n minutes ago)/Current volume

- DATA CLEANING

- Upon receiving features, we noticed significant outliers in many parts, so we created ranges for each component to include at least 99.9% of the data and constrained outliers to the ends of the contents.

- Also, we need to parse dates by using pandas to_datetime function

- Our output variable was then the predicted price ratio x minutes from the current timestep, where x =5,10,20.

- SPLITTING OF DATA

- We divided the data into train validation test set in ratio 60-20-20. As the data is Timeseries it is necessary to keep similar dates together without randomizing to make result generalizable.

- Basic Approach is:

- Classification: First to classify whether the price will increase or decrease.

- Regression: Then , if it is increasing ,by how much value it increased. Similarly If it is decreasing then by how much value.

- Model Implementation

- Weighted Logistic Regression

- Using a weighted logistic regression model based on the sign of the price change, our loss function would be accurately correlated to the gains we were making with our strategy of either investing at each time step or doing nothing. We predicted the sign of the price change by applying the indicator function on outputs and using the absolute value of the outputs as the individual weights. This way, the loss function that we are minimizing is calculated as below, and our logistic regression aims to minimize loss, which correlates to us maximizing our gains:

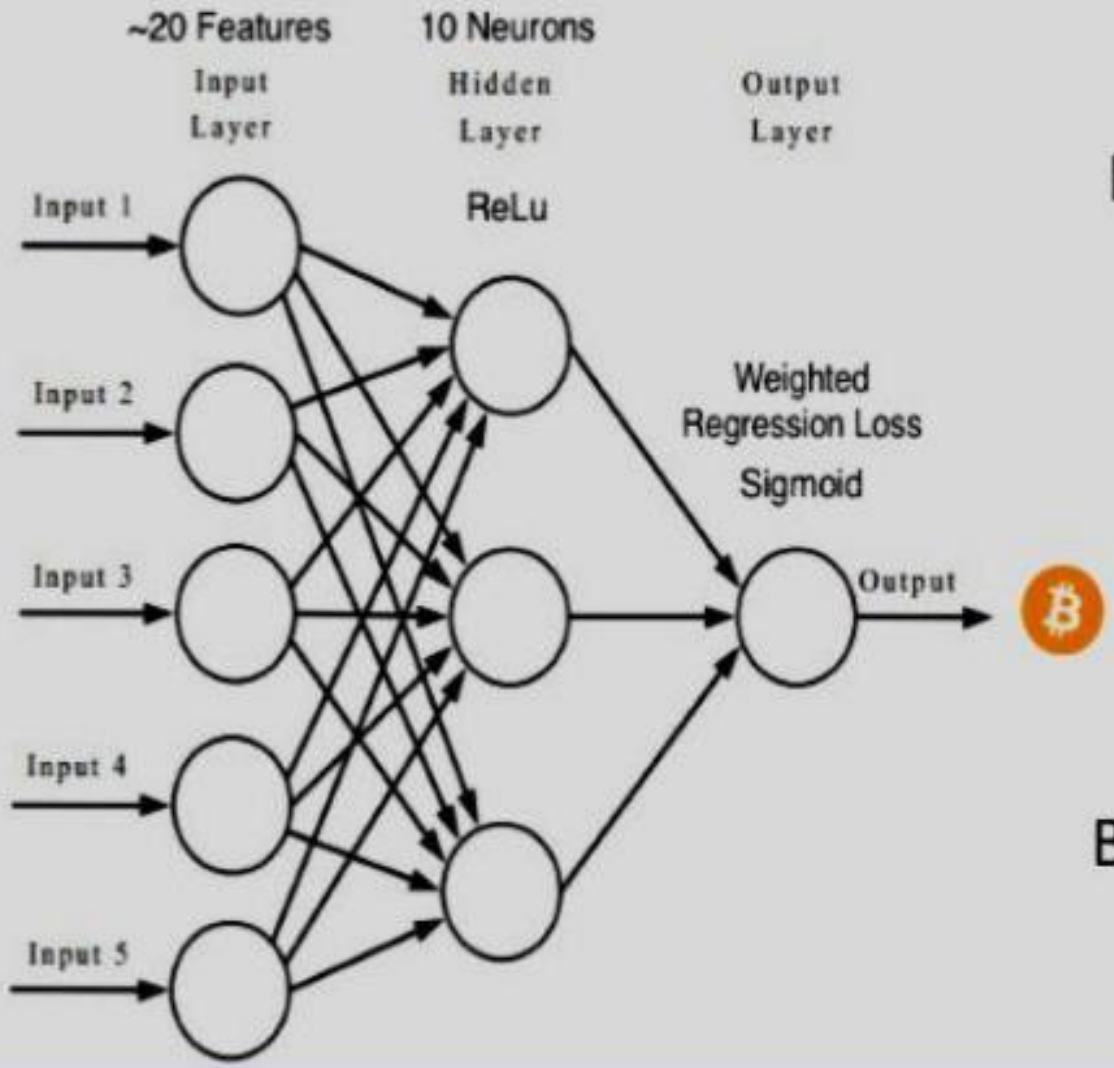
- The weighted accuracy is a weighted measurement to judge the accuracy of our algorithm in proportion to the gains that we're receiving. The gains correspond to the average price ratio increase we get each time step.

$$WA(\hat{y}, y) = \frac{\sum_i w_i \mathbb{1}[|\hat{y}_i - z_i| < 0.5]}{\sum_i w_i}$$

$$G(\hat{y}, y) = \sum_i y_i \mathbb{1}[\hat{y}_i > 0.5]$$

- PCA AND FEATURE SELECTION
- We decided to run PCA to check the principal components and their corresponding values of our feature matrix to evaluate which features had the largest impact on the variance of our data. First, we examined the correlation between variables by examining the eigenvectors of our covariance matrix. With that, we removed 3 classes of features (20 variables):
- WAP was consistently less varied than AP while being highly correlated
- VR consistently had low variance and was not a good predictor
- R was highly correlated A, so it was removed, as A captured more information We removed these 20 features from our data, and then ran PCA on our training set. We optimized the number of principal components to use in our data transformation by creating a model and maximizing the gains on our dev set. We consistently saw a jump in gains from 8 to 9 components, but found that using about 20 components optimized our gains

- . Neural network
- we create a neural network because it is suspected that the predictive model based on our features wasn’t strictly linear. We constructed a single hidden layer neural network with a rectifier (ReLU) activation function for the hidden layers and sigmoid activation for the output layer. The choice of sigmoid for the output layer was to mimic the logistic regression we were previously running, and we likewise utilized the same loss function such that we were optimizing for gains again. We chose ReLU because it is typically used for neural networks, as it is much faster at training than other non-linear functions, and our inputs were all real numbers. We then used mini-batch gradient descent. We divided the training data into 60 batches, and ran backpropagation on each batch. We did this for a total of 30 epochs. Our approach in terms of evaluation and convergence were the same, but we were instead utilizing a more complicated model that would be a better predictor for the data.



Forwards Propagation:

$$Z_1 = XW_1 + b_1$$

$$A_1 = \mathrm{ReLU}(Z_1)$$

$$Z_2 = A_1W_2 + b_2$$

$$\hat{y} = \sigma(Z_2)$$

Backwards Propagation:

$$\delta_2 = |y| \circ (\hat{y} - y)$$

$$\delta_1 = \delta_2 W_2^T \circ \mathbb{1}[A_1 > 0]$$

$$\nabla W_1 = X^T\delta_1 + 2\lambda W_1$$

$$\nabla b_1 = \delta_1$$

$$\nabla W_2 = A_1^T\delta_2 + 2\lambda W_2$$

$$\nabla b_2 = \delta_2$$

- RESULTS
- To evaluate our models, we used the following three metrics:
- Weighted Average (formula given in previous section)
- Gains (formula given in previous section)
- AUC: The area under the curve measuring the true positive rate vs. the false positive rate Gains was our primary measure of success, as it was indicative of how much money we would get for our trading strategy. The weighted accuracy was our secondary measure, which adjusted the classification accuracy based on the Weights that we used in our loss function Here are the results: All three of our models had gains that significantly outperformed the average increase per minute in bitcoin, suggesting that their usage on the market could result in larger gains than simply buying and holding bitcoin. Reducing the number of features and implementing PCA increased our average gains by a small, yet nontrivial, amount, while implementing the neural network significantly increased our gains. Both models beat out our basic weighted logistic regression, and performed significantly better than our second baseline based on analysis of price x minutes ago.