Using Bitcoin Data to Create a Profitable Algorithmic Trading Strategy

Bitcoin Trading Background

Bitcoin and other cryptocurrencies include a high amount of volatility in their markets, making algorithmic trading strategies an interesting problem to model.

Some of the most popular markets, (Coinbase, Bitfinex, and Bitstamp) all include fees for each trade made, making profitable high frequency trades quite difficult. GDAX combats these problems, as it does not include any transaction fees if we only make trades rather than taking them.

We scraped data from historical GDAX prices, which gave us parameters like open/close/high/low price and Volume in every one-minute time interval over the past year, which was around 450,000 data points.

The basis for our training strategy was to buy and then sell in x minutes if our model predicted Bitcoin to go up, and to do nothing otherwise. Thus, our intent was to predict the ratio of the price x minutes later to now.

Features and Outputs

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: Proportion of convex change every minute over last n minutes
- R: Ratio of price n minutes ago to 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

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

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

Baseline

Our primary baseline was the average increase in price over x minutes. It corresponded to what our gains would be if we always bought and sold in x minutes at every time step.

Our secondary baseline was based on looking at the ratio between the current price and the price x minutes ago, and investing only when that ratio was less than 1. This strategy ended up getting reasonable returns, but still lost by a fair amount to our three models.

Weighted Logistic Regression Model

Initially, we planned on doing a normal logistic regression, but quickly realized that such a model would not maximize our target. Therefore, we first made a weighted logistic classification model on the sign of the price change x minutes from the current time, where x = 5,10,20.

Our loss function became a weighted logistic regression loss function, where the weights were the absolute values of the output variable, so that our cost would negatively correspond to our gains based on our trading strategy. Thus, minimizing cost would correspond to high gains.

$$\mathcal{L}(\hat{y}, y) = \sum_i w_i(-(z_i \log(\hat{y_i}) + (1 - z_i)\log(1 - \hat{y_i})))$$

$$w_i = |y_i|, z_i = \mathbb{1}[y_i > 0]$$

Additionally, we defined our weighted accuracy and gains as

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

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.

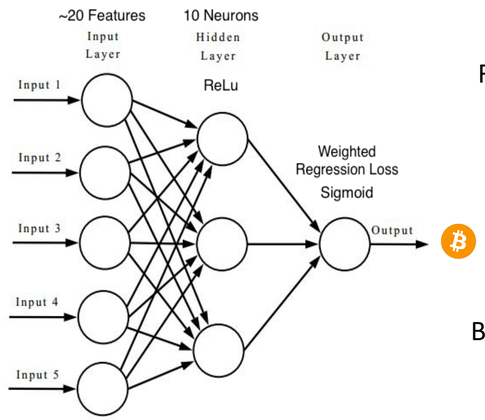
First, we examined the correlation between variables by examining the eigenvectors of our covariance matrix. With that, we removed 3 classes of features (20 variables):

- 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 repeated the weighted logistic classification model, except as a weighted neural network with the addition of a single hidden layer of neurons. Our output variable was the same as before, so we used the same loss function, as well as the same calculation of weighted accuracy and gains. We first ran PCA as described above, then transformed our data before running it through the neural network. Because the number of features after PCA could vary, we made our neural network vary the number of input features, though we did fix the number of neurons in the hidden layer at 10. We used 30 epochs, and divided our training data into 60 batches, each of size about 3,500. We implemented a regularization factor that varied for each output variable, but was in the range $[10^{-6}, 10^{-5}]$.



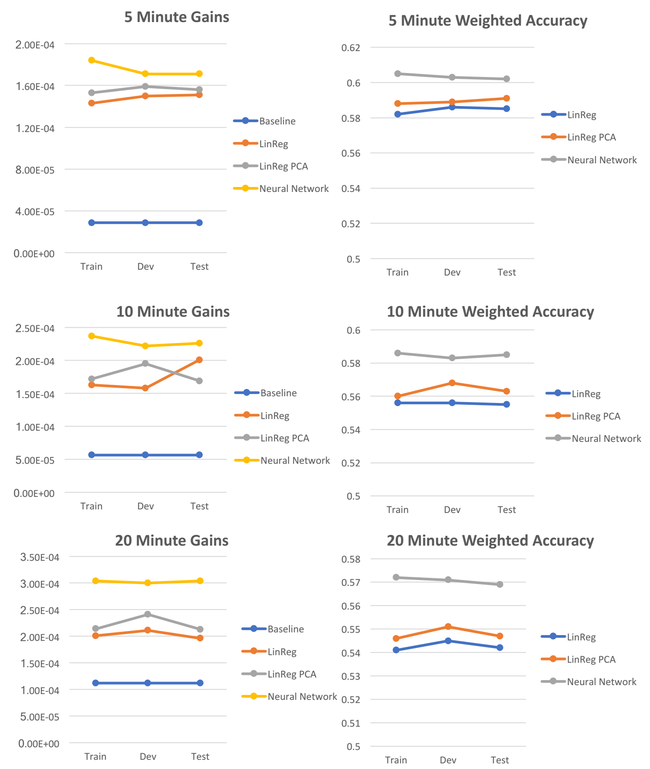
Forwards Propagation:

$$Z_1 = XW_1 + b_1$$
$$A_1 = \mathrm{ReLU}(Z_1)$$
$$Z_2 = A_1W_2 + b_2$$
$$\hat{y} = \sigma(Z_2)$$

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



Conclusions and Future Work

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 along with that 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.

A next step we want to take to validate our model is to run it on the current GDAX bitcoin data from the past 2 months, which we did not include in our dataset. Because bitcoin encountered a significant spike during this period of time, it would be interesting to check if our algorithm was still valid.

Furthermore, we are considering possibly implementing a recurrent neural network, as it would draw upon the time series dependency of data.

We want to research the implementation of an algorithm based on our model on GDAX to test our model in real time and possibly make significant returns.

[1] "Bitcoincharts | Charts", *Bitcoincharts.com*, 2017. [Online]. Available: https://bitcoincharts.com/charts
[2] Madan, Isaac, Shaurya Saluja, and Aojia Zhao. "Automated Bitcoin Trading via Machine Learning Algorithms." (n.d.): 1-6. CS229. Web.
[3] Shah, Devavrat, and Kang Zhang. "Bayesian Regression and Bitcoin." 2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton) (2014): 1-6. Web.