Using Bitcoin Pricing Data to Create a Profitable Algorithmic Trading Strategy

***Abstract*—Bitcoin, especially as of late, has been an incredibly high rising, although incredibly volatile, currency. Because of this, an algorithmic trading model that can make accurate predictions of short-term market trends can take take advantage of the spikes in the bitcoin market while avoiding the sharp decreases, allowing for substantial gains. In this project, we created an algorithm that would predict the price of bitcoin in x minutes relative to the current price, where $x \in \{5, 10, 20\}$. Our investment strategy would then be to repeatedly invest for x minutes if the algorithm stated that the price increased, and do nothing otherwise. We modeled this problem as a classification problem, where the two categories were based on whether the price increased or decreased. Three models were used: a simple logistic regression, a logistic regression after Principal Component Analysis, and a neural network with one hidden layer with a ReLU activation function. In all three cases, our loss function was a weighted logistic loss function. We found that each model did better than the previous one, where our metric was the expected amount of gains we would make in x minutes following our strategy. All three models significantly beat the baseline of the average increase in bitcoin price in x minutes, suggesting that their implementation might lead to a profitable trading algorithm.**

INTRODUCTION

The topic of cryptocurrencies has become extremely popular recently, and as cryptocurrencies have become more popular in the mainstream, some of their prices have greatly increased as well. The most popular cryptocurrency is Bitcoin, which has risen by over 600% in the 6 months from June 15, 2017 to December 15, 2017.[2]

One of the problems with Bitcoin and other cryptocurrencies is the high amount of volatility in their pricing, meaning that price can fluctuate significantly over a small period of time. While this is an issue when it comes to investing long-term in cryptocurrency, this also means that creating an algorithmic trading strategy for cryptocurrencies would be an interesting, and possibly profitable, problem to work on. In this project, we decided to focus purely on Bitcoin, as it was the cryptocurrency with the most data and the highest volume of trading..

Some of the most popular markets, such as Coinbase, Bitfinex, and Bitstamp, all include fees for each trade made, making profitable high frequency trades essentially impossible due to the low expected returns from each trade. GDAX, a less-common exchange, combats these problems, as it does not include any transaction fees if we only make trades rather than taking them. Making trades refers to the idea of putting a possible trade on the market, and taking trades refers to the idea of accepting a possible trade already on the market.

We scraped data from historical GDAX prices using bitcoincharts.com [1]. This website gave us the open price, the close price, the high price, the low price, and the volume traded in every one-minute time interval over the past year. This meant that we had about 450,000 data points.

Using this data, we created three classification models. The input to our algorithms is a set of numeric features created by examining the price data for bitcoin in the time interval $[t - 80, t]$ and deriving information from them, such as the average price in the past 40 minutes relative to the current price. For more information on the exact features, refer to the section on the Dataset and Features. We then use a weighted logistic regression model, a weighted logistic regression model with fewer features, and a neural network to output a predicted binary change, which predicts whether the price at time $t + x$ is greater than or less than the price at time t, where $x \in \{5, 10, 20\}$.

RELATED WORK

A. Bitcoin Predictive Classification Approaches

We read through some previous bitcoin pricing papers from past 229 projects and other scholarly sources. Madan, Saluja, and Zhao in [3], approached the problem from a classification perspective. First, they predicted the sign of price change in 1 day, and got great accuracy, around 98%, but when transitioning to 10 minute data, they noticed it was much more volatile, and only got around 55%, showing that predictive bitcoin pricing not to be a simple problem to solve. This classification approach was similar to ours, however instead of aiming to maximize accuracy, because we were implementing a simple investment strategy, we decided to maximize for gains instead and let accuracy be a secondary measure.

In Shah and Zhang's paper in [4], they used Bayesian regression to create a predictive cost for the cost in n minutes directly. They compared regular linear regression, lasso regression, and settled on utilizing bayesian regression and latent variables to find price ratios and used that as a predictive measure. This inspired us to also approach the price prediction from a ratio based perspective rather than as a purely price difference way. They saw a doubling of investment in around a 60 day period, which was reasonable.

B. Utilizing Latent Variables or Trend Data

Another paper referenced utilized metrics that assisted in generalized time series classification, which also seemed relevant to our problems, like trends in the twitter activity, like the paper by Chen, Nikolav, and Shah [5], which approached the problem from analyzing the latent variables, or Qian and Zheng in [7] approach in trying to predict market trends after the release of a financial report.

This made us consider implementing these types of variables, but decided on a more quantitative approach utilizing only the data we received in order to create a more generalized model, as well as to better understand the relationships that our numerical data could tell us, which seemed more in line with the vision of our project.

C. Generalized Stock Market Strategies

We also researched other academic papers to generalized stock market changes. Thakur, Vadpey, and Ayyar [6] analyzed the pros and cons of using different machine learning techniques, like linear regression, ridge, lasso, K-nearest neighbors, and boosting. Results showed incremental benefits from an 82% benchmark from logistic regression. However, when analyzing our problem, we noticed that bitcoin was much more volatile and the $\tilde{5}1\%$ benchmark that we saw from linear regression made some of these techniques not super relevant to our problem.

Dataset and Features

We initially had data for $408,960$ 1-minute intervals, spanning from January 1, 2017 to October 11, 2017, which came from [1]. For each minute t, we calculated the 66 following features:

- H_n: $\frac{\text{High price in interval } [t-n,t]}{\text{Price at time } t}$ for $n \in \{1,2,5,10,20,40,80\}$
- L_n: $\frac{\text{Low price in interval } [t-n,t]}{\text{Price at time } t}$ for $n \in \{1,2,5,10,20,40,80\}$
- A_n: $\frac{\text{Average price in interval } [t-n,t]}{\text{Price at time } t}$ for $n \in \{5,10,20,40,80\}$
- V_n: Volume of bitcoin traded in interval $[t-n,t]$ in BTC for $n \in \{1,2,5,10,20,40,80\}$
- P_n: Proportion of increases in price over each minute interval in interval $[t-n,t]$ for $n \in \{1,2,5,10,20,40,80\}$
- AC_n: Proportion of increases in change in price over each minute interval in interval $[t-n,t]$ for $n \in \{1,2,5,10,20,40,80\}$
- R_n: $\frac{\text{Price at time } t-n}{\text{Price at time } t}$ for $n \in \{1,2,5,10,20,40,80\}$
- WAP_n: $\frac{\text{Average price over interval } [t-2n,t-n]}{\text{Average price over interval } [t-n,t]}$ for $n \in \{1,2,5,10,20,40\}$
- AP_n: $\frac{\text{Average price over interval } [t-2n,t-n]}{\text{Price at time } t}$ for $n \in \{1,2,5,10,20,40\}$
- VR_n: $\frac{\text{Volume n minutes ago}}{\text{Current volume}}$ for $n \in \{1,2,5,10,20,40\}$

After the creation of these features, we realized that for most features, there were a small number of data points ($< 0.1\%$) for which the values of the feature were extreme outliers. As a result, we decided to create a cap on the values that each feature could take; our range for each feature included greater than 99.9% of the dataset in all cases. We then changed the outlier values to one of the endpoints of the range, depending on whether the outlier was less than or greater than the range.

For each minute t, we also calculated the 3 following output variables:

- RV_x: $\log(\frac{\text{Price at time } t+x}{\text{Price at time } t})$ for $x \in \{5,10,20\}$

Some of the data was stale and a copy of the previous interval's data, due to the fact that there were occasional 1-minute intervals where no trading had occurred or where GDAX had not recorded the data. As such, we created a freshness factor f for each time interval, where we set $f(t)$ to be the minimum k such that the time interval $t-k$ had fresh data (note that for the vast majority of the time, $f(t)=0$). We then removed data for which $\sum_{i=0}^{9} f(t-i) \geq 10$. After this cleaning, we also removed roughly 200 examples at the end of the dataset corresponding to the beginning of the year. We ended up with a total of 435300 data points.

Of this data, we divided the data into train/dev/test with the ratio $60/20/20$. Because the data was time-series, it was important to keep similar times together in each set to make our results generalizable, so this was done without randomizing the data; that is, the first 60% of the data became the training set, the next 20% the dev set, and the last 20% the test set. So, the training set had size 261300, and the dev set and test set had size 87100 each.

Note that in one of our models, we applied PCA to the training dataset. In this case, we applied PCA on the training dataset, and picked the first l principal components such that transforming the dev set to have exactly the l features created by multiplying the original dev set by the l principal components resulted in the highest possible gains.

I. Methods

A. Baselines

We decided to make our baseline measurement the average gains and accuracy that we received from choosing to always invest. Because Bitcoin is steadily rising in value, this baseline would measure the natural appreciation of Bitcoin, and the natural gains that we would receive by simply holding on to Bitcoin.

Our second baseline that we used was a simple heuristic that we manually calculated. We analyzed the sign of change in price from 10 minutes ago and compared it to the current price. If the price was trending up, we invested. This was essentially a one feature classification problem, and it was a reasonable metric for prediction that we wanted to beat out.

We initially approached this problem from a purely price predicting manner. We took our features and current price and directly predict the future price as ratio using a regular linear regression, meaning our loss function was the least squared difference from actual price ratio $5, 10, 20$ minutes from the

current time. However, due to the fact that prices fluctuated wildly, our preliminary tests showed grim results, as we could not even beat our baseline, and we weren't optimizing the metric that we intended, which was our total gains.

B. Weighted Logistic Regression

Thus, we decided to transition into a weighted logistic regression model based on the sign of the price change, so that our loss function would be accurately correlate to the gains we were making with our strategy of either investing at each time step of doing nothing.

We made a prediction of 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 minimizing loss, which correlates to us maximizing our gains:

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

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

After running the weighted regression, we defined a few metrics; namely, the weighted accuracy WA and gains, G. 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.

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

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

C. Principal Component Analysis

In our second model, we applied Principal Component Analysis (PCA) to the data in order to figure out the correlations between our features and remove the noise in our data set. First we normalized our data by subtracting the mean of each feature and dividing by the standard deviation of each feature from each training point.

We then created the covariance matrix

$$\frac{1}{m} \Sigma_{i=1}^{m} x_i x_i^\top$$

where m is the number of time steps we have in the training set, and x_i represented the features from the i^{th} training data point. To find the k principal components, we took the k eigenvectors corresponding to the k largest eigenvalues.

By looking at the most significant eigenvectors, we projected the relationships between our features and did some self selection to remove either low variance features, or highly correlated features. We decided on removing WAP, VR, and R as a result of our analysis.

Then, we projected each data point onto the principal components by taking the matrix product between the training points and the matrix of concatenated eigenvectors, which got us our new feature set, and once again ran weighted linear regression.

Given that v_i is our i^{th} principal component, our new features for our data would be as below for our k principal components and X is the m by p matrix containing all our m training examples and their p features:

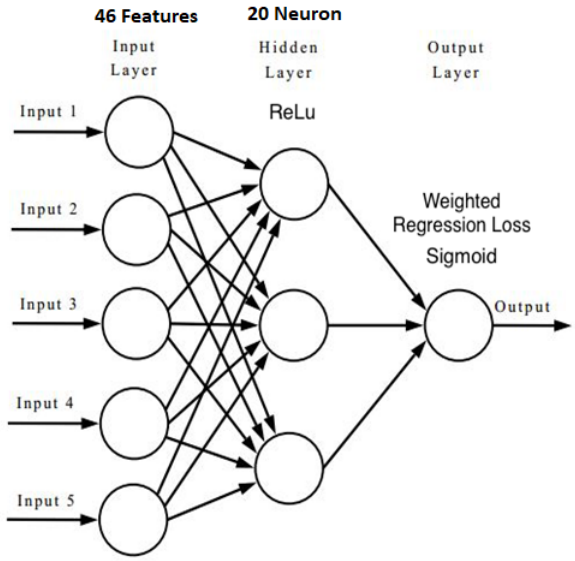
$$[Xv_1 \quad Xv_2 \quad \dots \quad Xv_k]$$

To find the optimal number of principal components to use, we utilized our development set in order to find the optimal k that maximized the gains for our development set.

D. Neural Network

We then decided to create a neural network because we 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.



Forward Propagation Formulas for our Neural Network:

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

Backward Propagation gradients for Neural Network:

$$\delta_2 = |y| \circ (\hat{y} - y)$$
$$\delta_1 = \delta_2 W_2^T \circ \mathbb{1}[A_1 > 0]$$
$$\nabla W_1 = X^T\delta_1 + 2\lambda W_1$$
$$\nabla b_1 = \delta_1$$
$$\nabla W_2 = A_1^T\delta_2 + 2\lambda W_2$$
$$\nabla b_2 = \delta_2$$

Results and Discussion

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:

Weighted Accuracy			
Model	$x = 5$	$x = 10$	$x = 20$
Logistic Regression	0.582	0.555	0.536
Logistic Regression with PCA	0.585	0.556	0.538
Neural Network	0.592	0.564	0.546

Gains			
Model	$x = 5$	$x = 10$	$x = 20$
Baseline	$3.06 \cdot 10^{-5}$	$6.20 \cdot 10^{-5}$	$1.25 \cdot 10^{-4}$
Logistic Regression	$1.51 \cdot 10^{-4}$	$1.67 \cdot 10^{-4}$	$1.95 \cdot 10^{-4}$
Logistic Regression with PCA	$1.56 \cdot 10^{-4}$	$1.71 \cdot 10^{-4}$	$2.02 \cdot 10^{-4}$
Neural Network	$1.68 \cdot 10^{-4}$	$1.91 \cdot 10^{-4}$	$2.31 \cdot 10^{-4}$

AUC			
Model	$x = 5$	$x = 10$	$x = 20$
Logistic Regression	0.592	0.563	0.541
Logistic Regression with PCA	0.599	0.571	0.548
Neural Network	0.623	0.592	0.565

The weighted logistic regression worked well as a model for giving us gains significantly higher than the baseline price increase of bitcoin. Our other two models showed incremental changes in weighted accuracy, gains, and AUC. Our PCA analysis worked as intended in removing some of the noise that was evident in our data, and also helped mitigate some of the variance present between our train and development set, as it allowed us to optimize the number of principal components in order to fit the development set.

The neural network performance metric consistently outperformed our other two metrics as seen in the tables. Our assumption that the model wasn't really linear was slightly helped by utilizing the non-linear activation functions of the neural network.

We overfit our data slightly to both our training and development set due to the fact that in a time series data, the later time steps weren't from exactly the same distribution as the first 80% that we got from the training and development data.

This was also evident because we used the train set to optimize our weights, and the development set to optimize our parameters, and was why our test set results were consistently worse than both training and development set metrics.

However, the intent of optimizing parameters for development set was to mitigate some of that over-fitting, and overall we still kept our metrics close, meaning our model wasn't a perfect generalized model for the price data, but it came close.

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 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.

CONTRIBUTIONS

Both of us did preliminary research into the bitcoin market and the potential ways to implement our algorithmic strategies. We also both formulated the features that we based upon our time data, and did most of the coding and analysis of our models together. The poster design and template was started by Justin and finishing edits were made by Dhruv.

Justin did further research onto current papers, and the approaches that some people had in tackling bitcoin and generalized stock market algorithmic trading. He also suggested using PCA to assist in feature selection and utilization of the development set, as well as remove noise from our linear model. Justin also did the data scraping from bitcoincharts to collect our data, as well as some preliminary excel algorithm analysis to parse features. He also researched into neural network packages to see what was in the realm of possibility of neural network frameworks before the final design.

Dhruv came up with the ideas for the trading strategy that would fit our machine learning framework, as well as the corresponding loss function and the gains metrics for our logistic regression models. He also performed some preliminary analysis in R to figure out which features mapped were great predictors by themselves, as well as looking at the shapes of all the output and input data, and decided on the cutoff points for the outlier data for each feature. For the neural network, he did the math for the propagation formulas to implement our batch gradient descent.

REFERENCES

[1] *Bitcoincharts.com*. (2017). Bitcoincharts Charts. [online] Available at: https://bitcoincharts.com/charts

[2] *Bitcoin Price Index - Real-time Bitcoin Price Charts.* [online] CoinDesk. Available at: https://www.coindesk.com/price/ [Accessed 16 Dec. 2017].

[3] Madan, Isaac, Shaurya Saluja, and Aojia Zhao. *Automated Bitcoin Trading via Machine Learning Algorithms.* (n.d.): 1-6. CS229. Web.

[4] Shah, Devavrat, and Kang Zhang. *Bayesian regression and Bitcoin.* arXiv preprint arXiv:1410.1231 (2014)

[5] G. H. Chen, S. Nikolov, and D. Shah, *A latent source model for nonparametric time series classification*, in Advances in Neural Information Processing Systems , pp. 10881096, 2013.

[6] Thakur, Saumitra, Theo Vadpey, and Sandeep Ayyar. *Predicting Stock Prices and Analyst Recommendations* (n.d.): 1-6. CS229. Web.

[7] Qian, Chen and Wenjie Zhang. *Stock Market Trends Prediction after Earning Release* (n.d.): 1-6. CS229. Web.