Comparitive Automated Bitcoin Trading Strategies

KAREEM HEGAZY and SAMUEL MUMFORD

1. INTRODUCTION

1.1 Bitcoin

Bitcoin is an international peer-to-peer traded crypto-currency which exhibits high volatility and is minimally impacted by current world events. Bitcoins are directly traded between individuals through intermediate sites that act as bitcoin markets. The largest of these markets is Bitstamp.net. In these markets people offer to buy or sell bitcoin at prices they define, or can put in a fill order to buy or sell at the best offered price. As bitcoin is traded around the world, unlike a stock or currency it is minimally affected by an individual corporation or the economic condition of a single country. This insulation from outside information makes bitcoin an ideal platform for trading through machine learning as the bitcoin price record itself should contain the vast majority of market information. Moreover, these data sets are free, while data sets for the stock market are not easily attainable. Finally, bitcoin price is highly volatile as seen in Figure 1, and thus small changes in algorithms can lead to large changes in algorithmic performance. We seek to use the relative insulation and high volatility of bitcoin to evaluate the trading performance of machine learning algorithms.

1.2 Our Approach and Previous Work

Using the procedures of modern machine learning trading papers, we compare seven different trading algorithms across multiple performance metrics in an attempt to address the unique challenges of bitcoin. Modern trading algorithms generally use reinforcement learning techniques to analyze markets[2][3][4]. However, it is unclear how similar the features of relatively stable stocks are to bitcoin. Our first task was thus applying reinforcement learning techniques to a new environment. Additionally, most papers analyze only one or two techniques for trading, often with similar architectures[5][2][3][4][7]. We instead seek to identify which general approaches work best by examining multiple classes of traders. Furthermore, many trading papers emphasize either trading performance or the accuracy without analyzing how well classification generalizes to trading[6][5][4]. By analyzing these two factors in conjunction we will gain a more nuanced understanding of how each algorithm works as a trader. Finally, past work often used small data sets[6][7] or unrealistically large time bins to make predictions more accurate[5]. We can make a more realistic reflection of the market and avoid overfitting by using smaller time bins and larger data sets. Through our comparative approach and knowledge of the bitcoin market, we hope to give a more nuanced analysis of algorithmic trading on a more realistically analyzed model of the bitcoin market.

1.3 Data Processing and Features

The bitcoin data that is readily found online provides a record of the time, quantity, and price of all bitcoin exchanged[1]. The data was processed by a weighted average of the bitcoin price within eight minute bins. Eight minutes was chosen so that bins would be large enough to reliably contain at least one trade, but small enough to capture high frequency fluctuations in bitcoin price. Such fluctuations to generally occur on timescales of ten minutes or longer[6][1].

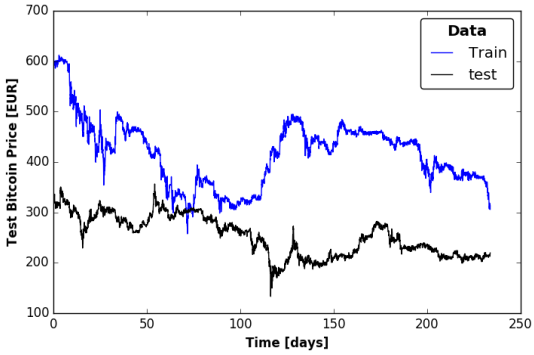

Fig. 1. The binned and smoothed bitcoin price data.

The binned price was also smoothed through locally weighted linear regression. The vector of prices for the previous 30 time bins before bin i was labeled P_i. The times of each of these bins was put in the form $(1, t_i)$ and concatenated to form a 2x30 matrix, T_i. Weights were calculated exponentially, with the j^{th} closest point to i receiving weight $w_j = \exp -(j^2)/\tau^2$ and $\tau = 3$[1]. The weight matrix W was then defined as diagonal with $W_{jj} = w_j$. The local linear fit offset and slope, S_i, was then fit by

$$S_i = (T_i^T W T_i)^{-1}(T_i^T W P_i) \tag{1}$$

and the price in bin i was replaced by $S_i(0) + t_i S_i(1)$. The final smoothed and binned profile of bitcoin price for our testing and training sets may be seen in Figure 1.

The first five left derivatives of bitcoin price were our primary features for algorithmic bitcoin trading. Left derivatives were used so as to not use information from "the future" to inform predictions of bitcoin price. Without binning and smoothing, high frequency noise in bitcoin price would have overwhelmed derivative methods. Many other features were considered, including the amount of bitcoin traded in a time bin, the product of the number of bitcoin traded and the derivatives of price, and the price of bitcoin itself, but including such features did not significantly change the performance of any of our algorithms. Additionally, adding in up to the first 15 left derivatives did not change the performance of our algorithms. Therefore, the first five left derivatives of price offered a computationally efficient and relatively rich feature set.

1.4 Trading Approach and Metrics

The general trading procedure of all but one of our algorithms was the same to allow us to compare trading performance. First, the algorithm is trained, yielding a set of training parameters θ. Next,

[1] Changing τ from 15 to 3 was what changed our results between the poster and the paper. The previous value overly smoothed price and made our predictions unrealistically reliable.

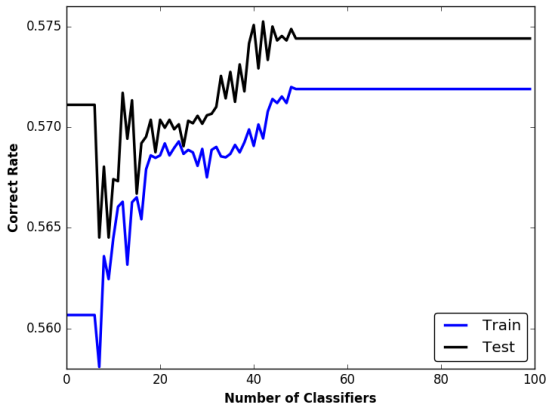

Fig. 2. Learning curve for stump boosting. Trading analysis was performed with 40 stumps.

for a point in the training or test set with features $x^{(i)}$, a function

$$q_i = q(x^{(i)}, \theta), \text{ with } -1 \leq q(x, \theta) \leq 1 \tag{2}$$

acts as a prediction of if price will go up or down in the next time bin. To trade based on this prediction in the i^{th} time bin, if $q_i > 0$, bitcoin is bought with q_i of all currently held cash. If $q_i < 0$, $|q_i|$ of all currently held bitcoin is sold. Such a procedure allows traders to have a sense of "momentum" or cumulative confidence in a trade, but also completely invest in or divest from bitcoin in one decision. All trading and classifying algorithms used were built from scratch in python other than the boosted decision tree method[8].

For a trained q and θ, we used three main performance metrics on the test and train set.

(1) Correct Rate (CR): The probability that the sign of q_i is correct in the training or testing set.
(2) Cumulative Weighted Confidence (CWC): The ratio of final asset value to initial asset value for trading without a trading fee.
(3) Profit: The ratio of final asset value to initial asset value for trading with a .25% fee applied to all bitcoins bought or sold.

These three metrics were selected to give a profile of the error rate, the significance of errors, and the trading performance of each algorithm.

2. ALGORITHMS EXAMINED

2.1 Weighted Linear Regression

The simplest algorithm implemented was weighted linear regression. Using our smoothing procedure, we found a local bitcoin price slope for each point, s_i. First finding the mean of the absolute value of slopes in the training set, s_0,

$$q_i = \frac{2}{1 + e^{-s_i/s_0}} - 1 \tag{3}$$

was used for algorithm evaluation.

2.2 Boosted Classifiers

To perform boosting on bitcoin price, one first defines a class of functions $\phi_j(x^{(i)})$ which return ± 1 based on the j^{th} set of function

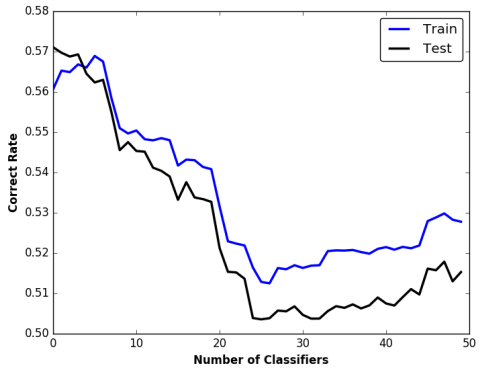

Fig. 3. Learning curve for tree boosting. Note the strong overfitting beginning after only one tree.

parameters and data features $x^{(i)}$. The goal of boosting is to create a composite classifying algorithm from the weighted sum of these ϕ_j according to weights θ, giving

$$h_\theta(x^{(i)}) = sign\left[\sum_{\theta_j} \theta_j \phi_j(x^{(i)})\right]. \tag{4}$$

In order to trade,

$$q_i = \frac{2}{1 + \exp\left(\sum_{\theta_j} \theta_j \phi_j(x^{(i)})\right)} - 1 \tag{5}$$

was used.

To make a boosted classifier, the list of θ_j is initially empty and $y^{(i)} = \pm 1$ indicates if the price of bitcoin increased or decreased in the next time bin. The boosting algorithm then proceeds as follows:

(1) Define weights for each point

$$w^i = \exp\left(-y^i \sum_{\theta_j} \theta_j \phi_j(x^{(i)})\right) \tag{6}$$

(2) Define a new classifier ϕ_u
(3) Define $W^+ = \sum_i w^i 1(\phi_u y^i > 0)$ and $W^- = \sum_i w^i 1(\phi_u y^i < 0)$. Add θ_u to the list of θs with

$$\theta_u = \frac{1}{2} \ln\left(\frac{-W^+}{W^-}\right). \tag{7}$$

This process is continued until minimum error in the test set is achieved.

Optimal decision stumps and decision trees were used as classifiers for boosting. As seen in Figures 2 and 3, the methodology of boosting did not work as expected for bitcoin trading. The boosting algorithms found one strong classifier instead of finding a series of weak classifiers and combining them. Correspondingly, only one tree was used before significant overfitting began to pose a problem and the CR did not improve significantly with more stumps added. This is likely caused by the fact that the first derivative is a much stronger classifier than the higher order derivatives and thus the boosting algorithms were dominated by the first stump or tree

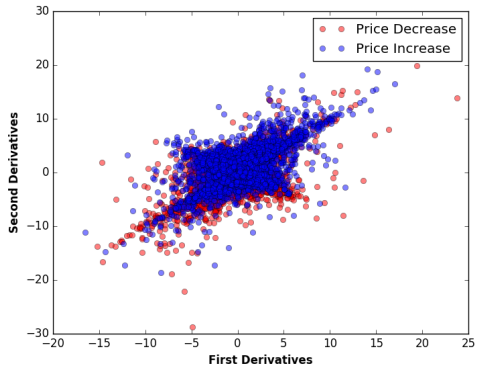

Fig. 4. Spread of first and second derivatives for use in GDA. Note that the price increase and decrease distributions overlap strongly and that there are some non-Gaussian features such as the elongated tails in the distribution.

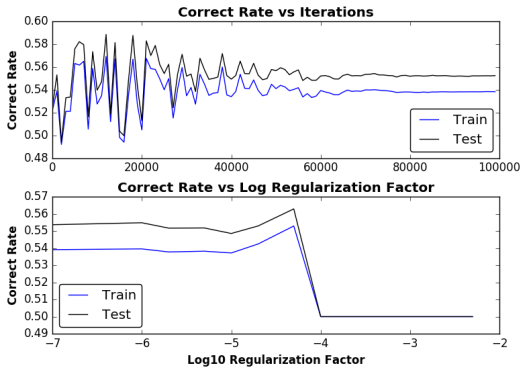

Fig. 5. Plots of the convergence in CR and CR vs. λ. CR did not change significantly with λ and appears to have converged well before the training iteration limit. The gap between training and testing CR also does not change significantly with iterations or regularisation factor, indicating that our feature set is small enough to avoid overfitting.

branch. In addition, adding the number of bitcoin traded in a bin as a feature did not mitigate this problem[2].

2.3 Gaussian Discriminant Analysis (GDA)

To perform GDA, first define $\sum_+$ and $\sum_-$ as the sums only over points where $y^{(i)} = \pm 1$. Next, construct two Gaussian distributions using the N data points with features x as

$$G_\pm(x) = \exp\left((x-\mu_\pm)^T\Sigma^{-1}(x-\mu_\pm)\right), \text{ where} \tag{8}$$

$$\mu_\pm = \frac{\sum_\pm x^{(i)}}{\sum_\pm 1}, \text{ and} \tag{9}$$

$$\Sigma = \frac{\sum_i x^{(i)}x^{(i)T}}{N} - \frac{\sum_+ \mu_+\mu_+^T + \sum_- \mu_-\mu_-^T}{N}. \tag{10}$$

Then to trade, define

$$q(x^{(i)}) = \frac{2G_+(x^{(i)})}{G_+(x^{(i)}) + G_-(x^{(i)})} - 1. \tag{11}$$

GDA should work as an effective classifier if the derivatives are Gaussian distributed within the $y^{(i)} = \pm 1$ categories. As seen in Figure 4, the two features with the highest discriminating power are not Gaussian distributed, but have long tails.

2.4 Logistic Regression

Logistic Regression is a fitting algorithm for $y^{(i)} \in \{0,1\}$ and fitting hypothesis

$$h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}. \tag{12}$$

Our training parameters θ are optimized through stochastic gradient descent on the cost function

$$J(\theta) = \frac{\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}{2} - \frac{\lambda\theta^2}{2} \tag{13}$$

[2]These strange learning curves may look like a bug, but plugging in the data set from Problem Set 2 yielded identical learning curves to the set solutions. Additionally, if the bin size or smoothing parameter is increased to make the derivatives less noisy the test set error rate approaches 0 but the train set error rate does not decrease.

with regularization factor λ. This gives update rule with learning rate α of

$$\theta^{t+1} = \theta^t - \alpha\left(h_{\theta^t}(x^{(i)}) - y^i\right)x^{(i)} - \lambda\theta^t. \tag{14}$$

Training was done over 1500000 points with an annealing learning rate and $\lambda = 0$ as seen in Figure 5. Finally, trading was performed by converting the fitting function to a number between -1 and 1 through

$$q(x^{(i)}) = 2h_\theta\left(x^{(i)}\right) - 1. \tag{15}$$

2.5 Recurrent Reinforcement Learning (RRL)

RRL was first used as a trading algorithm by Moody and Saffell in 1999[2]. We altered their approach for the specific case of bitcoin, which cannot be shorted and varies strongly on short timescales. A trader must decide the ratio of assets to invest in bitcoin to total assets at each timestep. This ratio is denoted by F_t, defined through

$$F_t\left(\theta; x^{(t)}\right) = \frac{1+\tanh\left(\theta^T x^{(t)}\right)}{2}. \tag{16}$$

Here $x^{(t)}$ are the features of the t^{th} time bin. Our wealth at time T is deterimined by multiplying the ratio of asset value for the t^{th} bin to the $(t-1)^{th}$ bin for all $t < T$. This ratio depends only on r_t = [Price at t]/[Price at (t - 1)] $- 1$, F_t and F_{t+1} or by

$$W_T = W_0\prod_{t=1}^{T}(1+R_T) \tag{17}$$

where

$$R_t = (1+F_{t-1}r_t)(1-\delta|F_t - F_{t-1}|) - 1. \tag{18}$$

We used this profit ratio as the reward for transferring from state $F_{t-1} \to F_t$.

The cost function maximized was the log of profit or

$$U(R_1, R_2, \cdots, R_T) = \sum_t^T \ln(1+R_t). \tag{19}$$

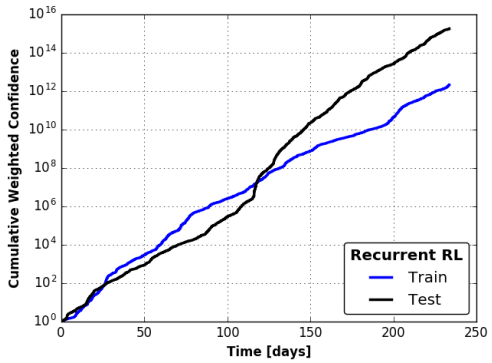

Fig. 6. CWC curve for RRL over the test and train sets. Note that the growth of the CWC appears to be roughly linear on a log scale, but with a slope broken into domains. The profit and CWC metrics therefore remain the best way to evaluate an algorithm trading, but their performance can be affected strongly by long-term trends in bitcoin price.

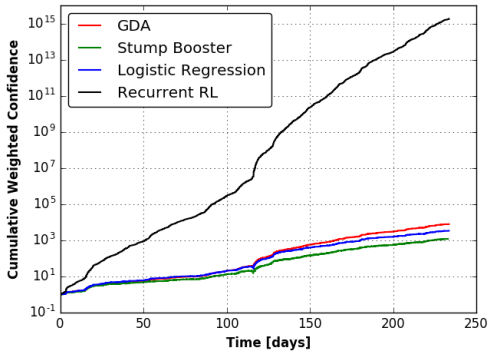

Fig. 7. Profiles of CWC in the testing set with time. Note the roughly exponential gain in capital how capital gains increase after the dip in bitcoin price at approximately 120 days.

Profit is maximized through online learning style gradient ascent. Since we analyze points starting from the oldest point,

$$\frac{dU_T(\theta)}{d\theta} = \sum_t^T \frac{dU_T}{dR_t}\left(\frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta}\right) \tag{20}$$

$$\rightarrow \frac{dU_t}{dR_t}\left(\frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta}\right) \tag{21}$$

can be expressed solely in terms of the most recent time step. The other derivatives required from the chain rule are

$$\frac{dR_t}{dF_t} = \beta\delta\left(1 + F_{t-1}r_t\right)\begin{cases}\beta = +1 & : F_t < F_{t-1}\\ \beta = -1 & : Otherwise\end{cases} \tag{22}$$

$$\frac{dR_t}{dF_{t-1}} = r_t\left(1 - \delta\left|F_t - F_{t-1}\right|\right) - \frac{dR_t}{dF_t} \tag{23}$$

$$\frac{dF_t}{d\theta} = \frac{\left(1 - (2F_t - 1)^2\right)}{2}x^{(t)} \tag{24}$$

As F_t is path dependent due to the fee, $\frac{dF_{t-1}}{d\theta}$ is the value calculated in the previous time step. RRL obtained highest train and test profit using only one run through the test data with a learning rate of 0.7. This procedure did not overfit, as is evident by the performance agreement between test and train data sets in Figure 6.

3. ANALYSIS: COMPARING ALGORITHM PERFORMANCE

As seen in Table 1, there is no evidence of significant overfitting for any of the trading algorithms. The CR in the test set was slightly higher than that of the train set for all algorithms. Test profits and CWCs were also higher than training results, implying that overfitting is unlikely. Additionally, the improved performance in the train set is likely caused by the relative stability of bitcoin price in the training data. It is possible that the data is underfit, but the logistic, stump, and tree boosting learning curves converged as seen in Figures 2, 3, and 5. The data therefore appears to be properly fit.

The simple weighted linear regression model performed poorly on all training metrics examined. First, the weighted linear model exhibits a CR of $< .5$, likely due to the high frequency fluctuations in bitcoin price seen in Figure 1. This poor classifier is also ineffective at trading. In trading both with and without a fee the weighted linear classifier loses almost all of the initial investment. The low accuracy of weighted linear prediction therefore demonstrates the difficulty of predicting bitcoin price.

The boosted classifying algorithms were the best binary predictors of bitcoin price change, but such predictions did not translate naturally to trading. The tree classifier accounts for correlations between different derivatives as well as the initial "stump" of the first derivative. Correspondingly, it has the highest CR. However, the tree booster was an ineffective trader because it only had one classifier. The trading decider q_i was therefore restricted to the values $2/(1 + \exp(\pm\theta_0)) - 1$. The tree booster hence has no sense of a confidence of a prediction, and the resulting trader simply bought or sold a set percentage of bitcoin. Without varying q_i effectively, the boosted tree algorithm appears to buy and sell too much bitcoin in each time bin and the mediocre CWC declines significantly when a fee is added.

The stump boosting method displayed similar trading behavior to the tree method. As seen in Figure 2, adding in more boosters did not significantly improve prediction accuracy. This behavior suggests that the boosted decision maker is dominated by the first stump and does not use many weak classifiers. The stump booster therefore acts much like a slightly worse version of the tree booster in terms of CR and CWC. The stump booster however performs better in trading with a fee than the tree booster does. This is likely due to a smaller θ_0 causing less bitcoin to be traded. A smaller θ_0 could also explain a lower stump CWC even if the stump and tree algorithms were to make identical predictions.

GDA and logistic regression performed poorly in terms of CR, but were relatively effective trading algorithms. If most predictions are made with $q_i \approx 0$, small changes in q_i would give different CRs, but would do little to change the performance when trading. Trading profit can be dominated by a few high-confidence buy or sell decisions, or by the values of q_i near ± 1. The q_i for logistic regression emerges naturally from the fit function and thus confidence should be built in. Logistic regression thus should form a more natural trading platform.

GDA surprisingly acts as a better trader than logistic regression. Viewing Figure 4, the derivative distribution does not appear to be entirely Gaussian. In such a case one would expect GDA performance to be bound by logistic regression. However, as the profit

Table I.

	Correct Rate		Cumulative Weighted Confidence		Profit	
	Train	Test	Train	Test	Train	Test
Weighted Linear	.4883		1.4015E-13		7.1480E-20	
Boosted Stumps	.5719	.5744	91.1972	1207.0390	.0391	1.4213
Boosted Trees	.5607	.5711	3.8887	7.6184	.1711	.5209
GDA	.5128	.5234	751.8358	7868.01246	3.8043	108.6978
Logistic Regression	.5372	.5504	828.4127	2809.0451	.2139	2.3755
RRL	.5436	.5561	2.2012E12	1.8174E15	12.4070	203.4374
Best Algorithm	Tree	Tree	RRL	RRL	RRL	RRL

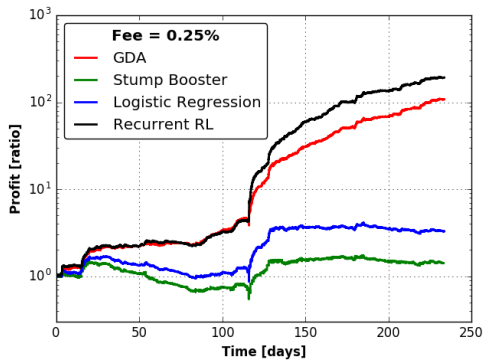

Fig. 8. Profile of profit with fee with time. Now RRL and GDA perform best, and are the only algorithms which appear to anticipate the price drop at 120 days. Additionally, GDA and RRL consistently make profit throughout the test set, with that rate of profit increasing after price begins to rise in the second half of the dataset. In contrast, nearly all the profit made by logistic regression and stump booster occurs in the short time period between 120 and 130 days when bitcoin price rose dramatically.

function is not the same function optimized by GDA or logistic regression, the trading performance of GDA is not bound by logistic regression. GDA performs similarly to logistic regression without a trading fee, but much better with a trading fee. The similar derivative profiles of the "price increase" and "price decrease" points may lead to GDA making fewer large $|q_i|$ trades. Correspondingly, GDA performance would decline less due to the fee.

The most effective trading algorithm both with and without a fee was RRL. Unlike all other algorithms tested, RRL directly maximizes profit. It correspondingly exhibits vastly superior trading performance despite having a CR commensurate with logistic regression in both the training and testing sets. Additionally, RRL trades best with a fee applied, likely because the fee can be incorporated into the optimization process itself instead of just being added on when trading. Finally, the ratio between CWC and profit is highest for RRL. This suggests that RRL when optimized without a fee makes very large buy/sell bets to take advantage of small local fluctuations, but changes behavior when trained with a fee. Such a change in behavior is indicative of the larger benefit offered by RRL. As RRL directly maximizes profit on our definition of how the market should act, market constraints can be incorporated naturally into the RRL process. In contrast, all other algorithms are trained independent of market constraints and then will simply trade differently.

In conclusion, viewing the profit results in Table 1, and Figures 7 and 8, both GDA and RRL were successful in generating large profits trading in our market model. These algorithms had a high ratio of final asset value to initial asset value despite the fact that bitcoin price declined in the testing and training data sets. Additionally, we selected efficacious features and were able to achieve higher CRs and profits than previous efforts to trade bitcoin[6][5] despite using more realistic time bin sizes[5]. In contrast, classifying methods did not work well as trading algorithms, despite having high CRs.

4. MOVING FORWARD

Although we traded on our simulated market effectively, there are many places where our approach could be improved upon. Primarily, it is unclear if we are effectively modeling the market behavior of the peers with whom we would be trading. Most concretely, the smoothing τ parameter was somewhat arbitrary. Without any smoothing, the derivative features become too noisy to have significant predictive power, and even RRL is not profitable. In contrast, with $\tau > 10$ the CRs for all algorithms other than weighted linear regression approach .8, and in particular GDA and logistic regression CR are $> .9$. However, with such a wide smoothing parameter our smoothed price would have little correspondence with the current actual price of bitcoin. Therefore, we would need to know over what time period an actual trader will "bin" or smooth bitcoin price to determine if our offered trade price is reasonable.

More abstractly, we did not account for how the market would react to us trading and always assumed that all trades we made would be instantly accepted. As we only had information on accepted trades, we had no information on market depth. In the case of dramatic changes in bitcoin price, many users would want to buy or sell at the same time, and yet we had to assume that any trades we offered would be accepted instantly, and not before the price either rose or fell. This clearly would not be true in the case of a crashing or bubbling. The assumption that we can instantly sell off all bitcoin before price drops also likely contributes to the overall absence of trade losses in Figures 7 and 8. Additionally, if we decided to sell all of our bitcoin at once after having multiplied our initial investment by a factor of around 200, we could flood the market with bitcoin and lower the price before it is bought. It is therefore unclear if our model of the market would hold for large investments or if our initial investment had grown significantly. The easiest way to address such questions would be to have a computer open continuously querying a bitcoin market for market depth information in real time. However, we could not do that in the time frame of our project.

REFERENCES

"bcEUR.csv.gz." *http://api.bitcoincharts.com/v1/csv/*. Bitcoincharts, n.d. Web. 16 Dec. 2016.

Moody, John, and Matthew Saffell. ”Reinforcement Learning for Trading.” *Advances in Neural Information Processing Systems* 11 (1999): 917-23. Web. 15 Dec. 2016.

Molina, Gabriel. ”Stock Trading with Recurrent Reinforcement Learning (RRL).” *CS229*, n.d. Web. 15 Dec. 2016.

Du, Xin, Jinjian Zhai, and Koupin Lv. ”Algorithm Trading Using Q-Learning and Recurrent Reinforcement Learning.” *CS229*, n.d. Web. 15 Dec. 2016.

Shah, Devavrat, and Kang Zhang. ”Bayesian Regression and Bitcoin.” 2014 *52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2014): 1-6. Web. 15 Dec. 2016.

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

Abikowski, Kamil. ”Application of Machine Learning Algorithms for Bitcoin Automated Trading.” *Studies in Big Data Machine Intelligence and Big Data in Industry* (2016): 161-68. Web.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. ”Scikit-learn: Machine Learning in Python.” Journal of Machine Learning Research 12 (2011): 2825-830. Web.