CS221 project: How to Succeed with Bitcoin Without Really Trying

Ellen Sebastian, Abaho Katabarwa, Guoxing Li

December 12, 2014

1 Introduction

Bitcoin is the world's first decentralized cryptocurrency. Since it was created by Satoshi Nakamoto in 2008, Bitcoin has become a popular international digital payment system because of its ability to bypass national financial regulatory bodies and stay resistant to the fluctuations of traditional currencies. Furthermore, the existence and volatility of bitcoin has created a vibrant and profitable electronic trading market. We would like to take advantage of various concepts in artificial intelligence and the vast amount of transactional/price data surrounding Bitcoin to create a Bitcoin trading bot whose goal will be to maximize profit by predicting future Bitcoin prices.

Using a neural network to predict Bitcoin prices and a simple rule-based action selector, our algorithm produces an average of 18% profit per month, or an annual interest rate of 617%. In one important case, when our algorithm is simulated between December 16, 2013 and November 10, 2014, a time period where the price of Bitcoin dropped from $950 to $320 [7], an initial cash investment of $1000 grows to $23797.

2 Related Work

Bitcoin is a rather new cryptocurrency. Therefore, few attempts have been made in trying to predict the price movement of bitcoin. However, due to its similar nature to stock trading, we assume that the methods used in forecasting stock price changes are applicable to predicting bitcoin price movements as well.

In [1], Shah, D. *et al.* propose a latent source model using Bayesian regression to solve Bitcoin price prediction problem, and achieved 89% return rate in two months' simulation. A lot of successes of predicting stock price fall in the category of utilizing artificial neural network. Kimoto, T. *et al.* develop a modular neural network to predict the best time to sell/buy stocks, which shows an excellent profit [3]. Kim, K. J. *et al.* apply genetic algorithms to feature discretization and the determination of connection weights for artificial neural networks (ANNs) to predict the stock price index [2]. Kamijo, K. I. *et al.* propose to use recurrent neural network to recognize common patterns in stock price movement [4].

3 Models and Algorithms

We split the problem of maximizing into two steps: price change prediction and action selection. Based on the outcome of price change prediction, we can choose an action that maximizes our profit.

Because long-term price prediction is a much harder problem than short-term prediction, we decided to frame the price-prediction step as follows: for any time t, use current and past data to predict Δp: the price of Bitcoin at $t + dt$. In this report, we will describe several algorithms for predicting Bitcoin price at $t + dt$. The success of this model depends on the fact that we can trade very frequently with little overhead cost; in fact, Bitcoin transaction fees are usually no more than 0.5% of the value traded [8], and it is possible to trade independently with no fees.

3.1 Price Prediction

3.1.1 Regression

Model:
We assume that Δp over some dt can be approximated by function of many real-valued features, including

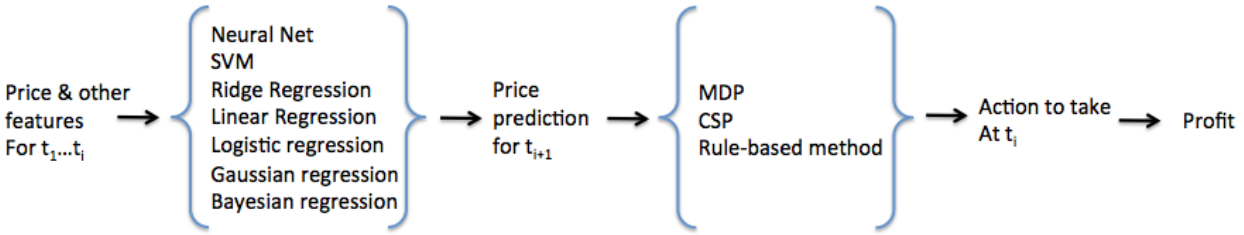

Figure 1: Schematic of the project split into two phases: price prediction and action selection. For price prediction we have tried many kinds of regression and Neural Nets. For action selection we tried using an MDP, CSP, and rule-based method.

previous prices and the log proportion of buy to sell transactions. For example, our feature vector x = [price 60 seconds ago, price 120 seconds ago, log buy/sell ratio, transaction frequency]. We use regression algorithms to fit our training data.

Algorithm:
We choose 1000 random historical time points for which we have feature data. For each time point, we accumulate a vector of features leading up to November 11, 2014, such as the average Δp per minute over the last 60 minutes, the same per day over the last 60 days, and log proportion of buy to sell transactions. Using the same input features, we try different regression algorithsm including Linear Regression, Ridge Regression, and SVM Regression. We also tried Gaussian Processes Regression. The use of this specific regression algorithm will be explained in later section.

3.1.2 Bayesian Regression

This approach is based on Shah, D. and Zhang, K.'s work in [1]. Bayesian regression refers to utilizing empirical data as proxy to perform Bayesian inference. Shah and Zhang proposed a latent source model using Bayesian regression for predicting Bitcoin price changes. Based on the assumption that price movements follow a set of patterns, one can use past price movements to predict future returns to some extent [5]. Studies also found that some geometric patterns such as heads-and-shoulders and triangle can be used to predict price changes.

$$\hat{y} = \frac{\sum_{i=1}^{n} y_i \exp\left(-\frac{1}{4}||x - x_i^2||_2^2\right)}{\sum_{i=1}^{n} \exp\left(-\frac{1}{4}||x - x_i^2||_2^2\right)} \tag{1}$$

Equation (1) shows an estimation on price change using Bayesian inference, which is used to construct features in our regression problem. To take into consideration the price movement patterns of different time frame, we calculate the price change for each time step, then generate time-series data of three different lengths: S_1 of time-length 30 minutes, S_2 of time-length 60 minutes, S_3 of time-length 120 minutes. For a single time step, each time-series will generate an estimate using equation (1), in which case $\hat{y}$ is the estimation of the price change at the next time step, x is a m dimensional vector consisting of price changes of previous m time steps, and n is the number of datapoints we use to make the estimation. We denote the estimated price change for each time-series as Δp^j where $1 \leq j \leq 3$. The final estimation Δp is produced as

$$\Delta p = w_0 + \sum_{j=1}^{3} w_j \Delta p^j \tag{2}$$

where $\mathbf{w} = (w_0, ..., w_3)$ are learnt parameters. We didn't include trading volumn data in equation (2) as they did in [1] since the order book data is not available to us. However, this shouldn't have a huge effect since the major assumption of this framework is that price movements can be predicted by identifying common patterns. To train our regression model, we divide the time-series price data into three equal sized periods. We use the first period to derive time series $S_j, 1 \leq j \leq 3$. The second period is used to learn $\mathbf{w}$ using linear regression. The last period is used to test our prediction. Since the dataset is fairly large, for each

time-series, we run k-means clustering with 100 clusters on x's. We further select 20 most effective clusters based on the standard deviatioin of all the y's in the cluster (lower standard deviation means predictions are more reliable), and pick the most representative pattern from each of those clusters (smallest distance to mean). This reduces n in equation (1) to 20 for each time series, which drastically improves the runtime.

3.1.3 Artificial Neural Networks

Model:
We modeled the problem as a time series prediction. Given a price data (the price data that will be fed into the artificial neural net can either be price, the percentage change in price, or the percent change or the percentage change in price), $P = [p = 0, p = 1, ..., p = n]$ chronologically arranged from $t = 0$ to n, we can create feature vectors $a = [p_1...p_k]$ where $1 \leq k \leq n - 1$ at regular intervals of P and target values $b = p_{k+1}$.

Algorithm:
Before beginning the algorithm we smooth the price data using Savitzky-Golay smoothing [6]. An illustrative example of this is provided below (Figure 2).

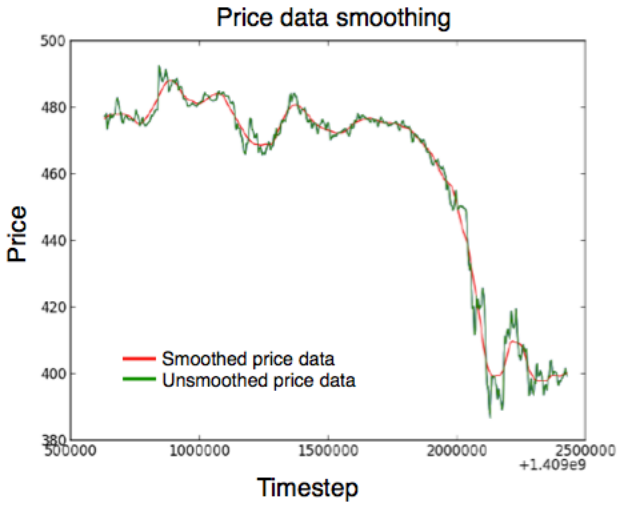

Figure 2: Price data smoothing. Before constructing the Neural Net, we smooth the price data using Savitzky-Golay cubic smoothing to remove excess jaggedness.

Now we formulate our neural network N. Given price data $P = [p = 0, p = 1, ..., p = n]$, we determine a window size of $k << |P|$. We then move this window along some interval of P while at each at each iteration using the window to create a feature vector and a target value. Once we have K feature vectors and target values we can train N and predict p_{k+1} given a feature vector $[p_0...p_k]$. Figure 3 shows a diagram to illustrate how the algorithm works.

Example:
Consider the price data $[.5, .6, .2, .1, 0.01, 0.2]$. We choose a window size of 2 and create the following feature vectors and target pairs $A = [([.5, .6], [.6])([.6, .2], [.1])([.2, .1], [0.01])([.1, 0.01], [0.2])]$. We can use A to train the neural network N. In order to predict the next data point we input the last feature vector $[0.01, 0.2]$ into N.

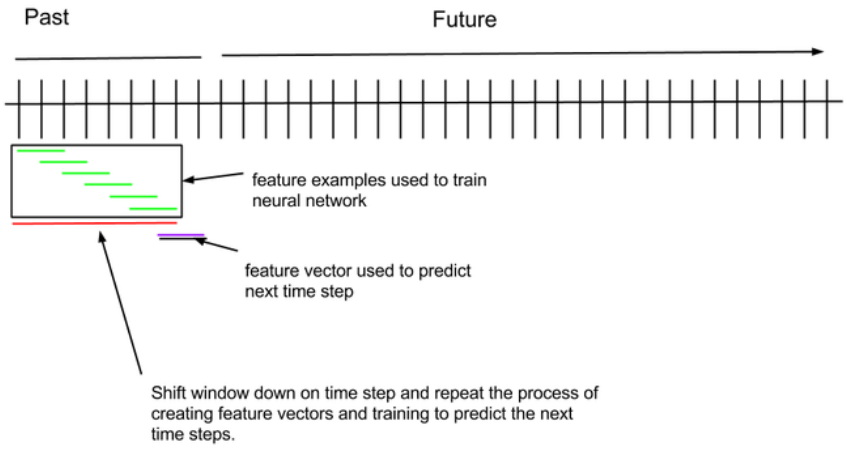

Figure 3: Diagram of Neural Net construction.

3.2 Action selection

3.2.1 Policy-based approach

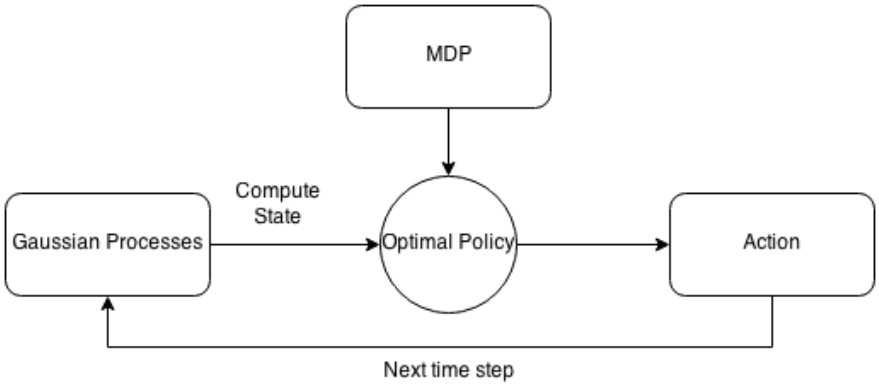

Figure 4: The flow of the GP + MDP framework

In this algorithm, we use Gaussian Processes (GP) and Markov Decision Process (MDP) to model our bitcoin trading problem. The basic idea is to use GP to predict a normal distribution of bitcoin price at the next time step, which can be used to estimate the transition probabilities in our MDP model. We first construct the MDP model based on the following simplified problem. We are given B bitcoins. Our task is to sell all of them within a fixed time period T, and maximize the revenue we received. Also assume that when we place an order at time t, all of the shares placed will be sold at the average price of that time immediately.

MDP Specifications:

- States: $(t, b, \Delta p, \sigma)$
  In the state representation, t means time steps remaining, b means number of bitcoins remaining, Δp means the predicted price change at the next time step, σ means the standard deviation of predicted price. To limit our state space, we discretize each element in our state space. Depending on how large we want our state space to be, we can choose the resolution beforehand. For predicted price, we can define our range by looking at the real price range in the time period that we are interested in. Based on different time step resolution, we adjust the scale of standard deviation accordingly.

- Actions: $\{$sell i bitcoins $(0 \leq i < b)\}$
  The resolution of bitcoin is the same as the one we chose for state space.

- Rewards(s, a, s): $-a \times s[\Delta p]$
  Rewards are calculated by the amount of bitcoins we chose to sell multiplied by negative predicted price difference (selling bitcoins at the current time step when predicting the price will go up will incur a negative reward).

- Transition probabilities: $T(s, a, s') \sim N(s[\Delta p], s[\sigma]^2)$
  The probability of transition from a state to a new state follows the normal distribution whose mean is $s[\Delta p]$, standard deviation is $s[\sigma]$. Since we are discretizing the states, we take the probability density and normalize it to give us an estimate on the transition probability. It's worth noting that since we don't have information on the transition probability with regard to predicted standard deviation, we assume it stays the same during state transitions.

- IsEnd(s): $s[t] = 0$
  The simulation ends when time runs out.

The framework of this approach is depicted in Figure 4. After constructing the MDP model, we run value iteration on a specific MDP problem by fixing parameters (resolution, price range, etc.), which outputs an optimal policy. When running the simulation, we use GP to generate predictions on the price change as well as its standard deviation at the next time step, which is used to construct a current state and suggests an action based on the optimal policy.

3.2.2 Variable-based approach

We further explore to model the trading problem as a Constraint Satisfaction problem. As a first step, we implemented a CSP that finds the optimal times to sell Bitcoins, but does not consider buying Bitcoins.

- *Variables*: Number of bitcoins to sell at each timestamp. There is one variable for each predicted price over a set period of time.

- *Potential*: Do not sell more Bitcoins than you own. Assignments that sell more Bitcoins than you own are assigned a weight of 0.

- *Potential*: Do not sell Bitcoins at a loss. Assignments that sell Bitcoins at a lower price than they were bought receive a weight of 0.

- *Potential*: timestamps with more certain (earlier) predictions are weighted higher: $\frac{1}{\log(ts+2)}$

- *Potential*: timestamps with higher predicted prices have higher weight=max(predicted profit,1)

At every timestep, we recalculated price predictions and used the price predictions to construct a new CSP. However, the above CSP is limited to selling but not buying Bitcoins. We next implemented a second CSP that decides whether to buy, sell, or do nothing at a single time. It is defined as follows:

- *Variable*: What action to take a the current timestamp: buy 1 Bitcoin, sell 1 Bitcoin, or do nothing.

- *Potential*: Do not sell Bitcoins at a loss. Assignments that sell Bitcoins at a lower price than they were bought receive a weight of 0. This potential contributes a weight of 1 to assignments that sell Bitcoins at a higher price than they were bought.

- *Potential*: Do not sell Bitcoins at a loss. Assignments that sell Bitcoins at a lower price than they were bought receive a weight of 0.

- *Potential*: Do not own more Bitcoins than a given maximum number. If the maximum limit of Bitcoins to own has been reached, assignments selling an additional Bitcoin will receive a weight of 0.

- *Potential*: Buy Bitcoins at predicted troughs in price. Assignments that buy Bitcoins before a predicted decrease in price receive a weight of 0. Assignments that sell Bitcoins before a predicted increase in price have weight proportional to the strength of the downward trend before the current timestamp.

- *Potential*: Sell Bitcoins at predicted peaks in price. Assignments that sell Bitcoins before a predicted increase in price receive a weight of 0. Assignments that sell Bitcoins before a predicted decrease in price have weight proportional to the strength of the upward trend before the current timestamp.

3.2.3 Rule-based approach

While building the CSP, we recognized that our price predictions are not accurate more than 1 step in the future. If we can only decide based on one time step, the problem is simplified into assigning an action to a single time step. In this case, a CSP would only have one variable. It is more flexible and straightforward to assign an action to a single time step through a simple rule-based approach than through a CSP. The optimal trading strategy given perfect price prediction would be always sell all of your bitcoins at peaks and always buy with all of your money at thoughs. However, it is impossible to reliably detect peaks and troughs without having accurate price predictions far into the future. Therefore, the rule-based trading strategy we used is more conservative, selling and buying only one Bitcoin at a time. We sell 1 bitcoin before predicted decreases in price and buy 1 bitcoin before predicted increases in price. We formualte the rules in the following equation

$$\text{Action} = \begin{cases} \text{Sell 1 bitcoin} & \text{if } \Delta p \geq c, b > 0 \\ \text{Buy 1 bitcoin} & \text{if } \Delta p \leq -c, w \geq p \end{cases} \tag{3}$$

where Δp is the predicted price change, c is a threshold we set, b is the number of bitcoins we currently have, w is the amount of money in hand, and p is the current bitcoin price.

4 Experiments and Results

4.1 Data

The data we used majorly come from two sources, BlockChain [10] and OKCoin [**?**]. BlockChain offers us daily data on market price, number of transactions, total volumn, and mining difficulty etc. OKCoin offers price data of finer granularity. We collected per minute price data from July 2010 to November 2014 from OKCoin. We preprocessed the price data to calculate the price change over different time step.

4.2 Price Prediction

4.2.1 Evaluation Metric

Though price prediction is a regression problem, we chose not to use Root Mean Square Error (RMSE) for two reasons. First, since the time step resolution we chose is small (1 minute to 1 hour), the price movement is always too small to make RMSE meaningful. Also, during trading it makes sense to look at the direction of price change instead of absolute value change. Therefore, we decided to use accuracy of prediction direction as our evaluation metric. Accuracy is defined as the number of correct direction predictions over all predictions.

4.2.2 Prediction Performance Comparison

We first ran preliminary experiments on selecting features, and found that non-price features didn't add value in improving predictions at all. This makes sense since most of the non-price features we tried are correlated with price movement rather than a forecasting signal. Therefore, we decided to use only price data when experimenting with different prediction algorithms.

To fully understand the performance of different algorithms, we use the per minute price data from OKCoin and picked a period consisting 10000 data points to run experiement on. For regular regression algorithms, we ran 10-fold cross-validation on the dataset. For Bayesian regression, as mentioned in the algorithm description, we divided the data into three equal-sized periods for collecting time series, training, and testing. For neural network related methods, we used first two thirds of data to train, and the last third of data to run test. The result is shown in Figure 5.

From Figure 5, we can see that neural network with smoothing performed the best ($\sim 60\%$ Accuracy). All methods of regression performed poorly in predicting Δp. Accuracy is no better than random in all cases; that is, Accuracy ≈ 0.5. We hypothesize that this is because Bitcoin prices cannot meaningfully be fit to a function curve using the features we selected. For neural network, by using Savitzky-Golay smoothing and a forward-feed neural net with a window size of 24 time steps, we were able to consistently acheive 60% accurate directionality (ergo, we were able to predict whether it was a positive or negative change in

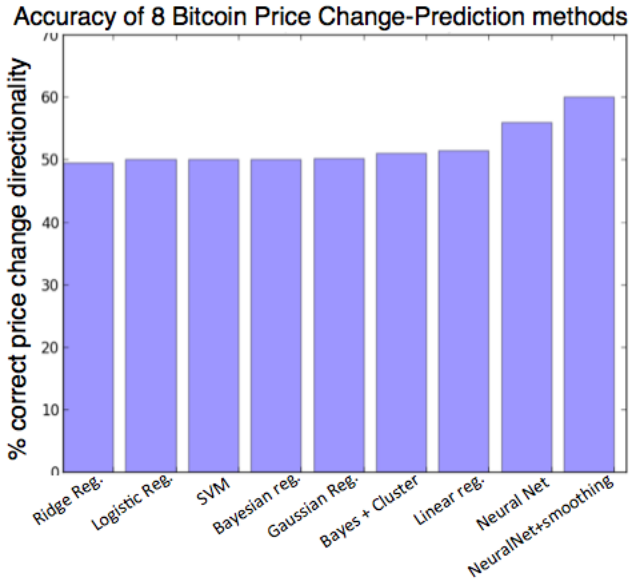

Figure 5: Accuracy of 8 Bitcoin price change prediction methods. All regression algorithms achieve approximately 0.5 accuracy. Neural Net with Smoothing achieves 60% accuracy.

price 60% of the time) of price predictions. Figure 6 and figure 7 show the predicted price change error and predicted price error respectively. It's worth mentioning that even though Bayesian regression didn't achieve good accuracy, it was still able to identify some common patterns (Figure 8). Also, we couldn't compare our implementation with Shah *et al.*'s work since they didn't present their price accuracy result (they only showed their simulation result of running a basic strategy based on predicted price).

4.3 Trading

4.3.1 Evaluation Metric

We simply use return rate as the primary evaluation metric. Return rate is defined as follows

$$V(t) = b(t) \times p(t) + c(t) \tag{4}$$

$$r = \frac{V(T) - V(0)}{V(0)} \tag{5}$$

where $V(t)$ denotes the total value at time step t, $b(t)$, $p(t)$, $c(t)$ is the number of bitcoins, price of bitcoin, and cash in hand at time t, r denotes return rate, $V(T)$ is the final value after simulation, and $V(0)$ is the inital value.

4.3.2 Policy-based approach

The MDP model we formulated in section 3.2.1 is based on a simplified problem where one can only sell bitcoins during trading. So we experimented MDP separately. We randomly chose time periods of 2 hours in November 2014 to experiment with MDP model. To compare the effectiveness of our MDP model, we run three approaches side by side, which are MDP learned policy running on Gaussian Process predicted price and standard deviation (GP), MDP learned policy running on perfect price prediction (Perfect), and selling all at the highest price possible (Max). In all tests, we picked initial number of bitcoins to be 10, price resolution to be \$0.1, and maximum standard deviation to be \$0.5. The result is shown in Table 1.

We can see that the profit MDP generates is far from maximum profit. There are several critical problems with our MDP model as we discovered while running the tests. First of all, running value iteration on MDP takes a long time due to numerous states we have. From Table 1, we can observe that even a simulation of 2 hours generated 166,320 states, which takes 6 minutes for value iteration to converge. The worse part is

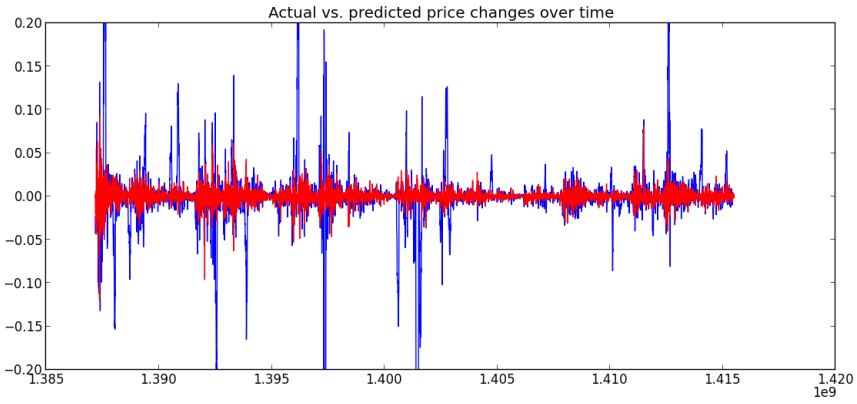

Figure 6: Actual (red) and predicted (blue) percent changes in price for each timestamp between December 16, 2013 and November 10, 2014.

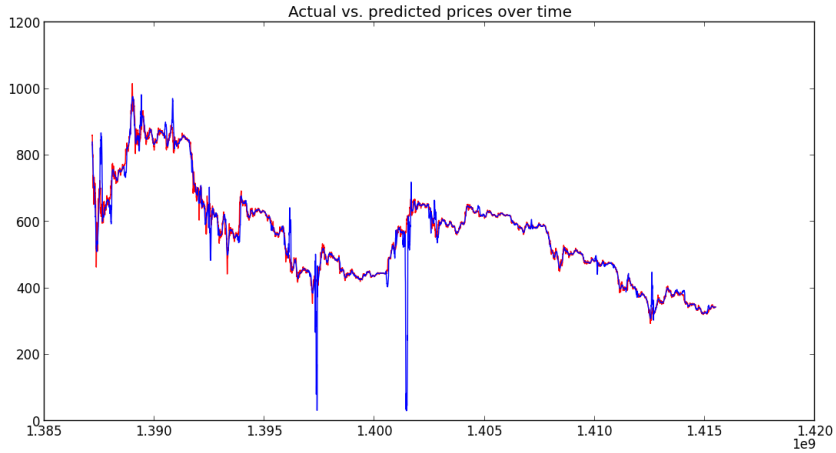

Figure 7: Actual (red) and predicted (blue) prices for each timestamp between December 16, 2013 and November 10, 2014. Note that the predicted prices are often a smoothed version of the actual prices, since predictions come from smoothed data.

that runtime seems to grow exponentially with increasing simulation time since longer simulation increases price range. Secondly, the state we constructed doesn't take current price into consideration, which is an important factor in derermining whether a state is good or bad. However, if we add current price into the state representation, not only do we have a much large state space, but we also have no idea on the transition probability of predicted price change. We believe these are the major reasons that make our MDP algorithm behave so poorly.

4.3.3 Variable-based approach

The first CSP described in section 3.2.2 also only considers selling Bitcoins. We found that, given perfect price predictions, the CSP would find the optimal assignent of Bitcoin sellings. However, this success is not very useful because it relies on far-future price predictions and is not independent: it requires a human to buy Bitcoins.

The second CSP described in section 3.2.2 both buys and sells Bitcoins. We tested the algorithm randomly selecting 100 starting points between July 18, 2010 and November 10, 2014. For each starting point, we simulated using the CSP for 24 hour-long time periods.

This CSP usually produces a small amount of profit, about 1.5% return rate over 1440 minutes (1 day)

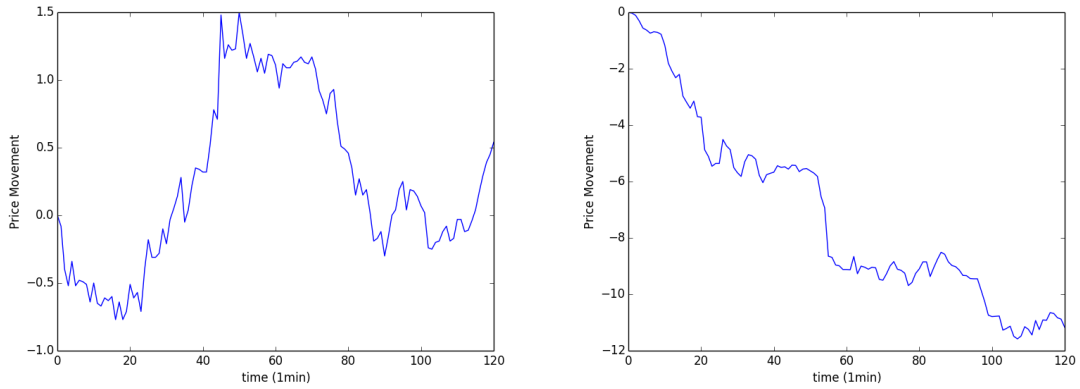

Figure 8: Common patterns identified by Bayesian regression framework. Head-and-shoulds (left) and Descending Triangle (right).

Simulation Time (min)	**Price Range**	**Number of States**	**Runtime (min)**	**GP**	**Perfect**	**Max**
120	[-1.0, 1.0]	166320	6	+0.016%	+0.02%	+1.68%
180	[-2.0, 2.0]	487080	22	+0.00%	0.01%	0.56%

Table 1: Performance of MDP model

or 1440 hours. We observed the issue that the algorithm often lost money due to selling too early, e.g. selling before reaching a peak or buying before reaching a trough. This meant that the profit was bounded even with perfect price predictions. For example, in a controlled test case where the price decreased from $20 to $10 by $1 each minute, the algorithm bought 1 bitcoin each minute after observing the trend of decreasing prices. As a consequence, our return rate was -46%, since we invested more than the final value of our bitcoin assets.

4.3.4 Rule-based approach

The rule-based approach was extremely successful in producing high profits over a variety of price conditions, time constraints, and spending allowances. We focused on the time period December 16, 2013-November 10, 2014. This period started directly after the all-time price spike in December 2013, resulting from a steep increase from $120 to over $1000 in November 2013. If we started testing before the spike, all results would trivially produce profit due to the overall increase in Bitcoin price.

In order to assess the impact that our poor price predictions had on our profit, we performed all testing on two sets of price predictions:

1. Perfect price predictions generated by looking ahead at the actual prices for the next time step. This allows us to test the efficacy of the rule-based action-selection method independently from price prediction.

2. 60% directionality-accurate price predictions from Neural Net with smoothing. This allows us to assess the quality of the complete algorithm.

We used return rate as described above to evaluate the performance of this rule-based approach, and conducted the experiment as follows. First, we chose 100 random starting time points between July 18, 2010 and November 10, 2014. For each starting point, we simulated using the CSP for 720 hour-long time periods (amounting to 30 days). We found that monthly profit was around 100% using perfect price predictions, and about 18% using Neural Net predictions. Profit varies according to the parameters used. Profit is maxmized when a large amount of cash investment is allowed and when there is no limit to the number it Bitcoins that can be owned at once.

We also simulated running our rule-based trading algorithm for 11 months between December 16, 2013 and November 10, 2013. We found that up to 4060% profit or $33,466 could be gained using Neural Net price predictions. We found that it is more profitable to trade on a 5-minute basis than an hourly basis. However, it was computationally prohibitive to explore the effects of shorter timesteps.

c(t=0)	**b(t=0)**	**BTC_lim**	**Timestep**	$\mathbf{V_{NN}(t=T)}$	$\mathbf{r_{NN}}$	$\mathbf{V_{Perf}(t=T)}$	$\mathbf{r_{Perf}}$
10000	0	10	1 hour	28896.29	2889%	30781.66	**3080%**
10000	0	100	1 hour	28185.07	282%	33466.06	**334%**
1000	0	100	1 hour	14365.52	1436%	21890.99	**2189%**
1000	0	5	1 hour	8915.97	891%	18834.24	**1883%**
1000	0	1	1 hour	10155.28	1049%	2811.67	**2810%**
500	0	100	1 hour	20149.92	**4030%**	5645.05	1130%
500	5	100	1 hour	27698.11	**577%**	23838.06	449%
1000	0	100	5 minutes	23454.42	**2379%**	N/A	N/A

Table 2: Return rate from running Neural Net and Rule-based trading decision algorithm with varying parameters. The first two columns refer to the user's capital at the beginning of the algorithm: the number of dollars and bitcoins. "BTC_lim" refers to the maximum number of Bitcoins the user can own at any time. Timestep refers to how often a decision is made. The final 4 columns list the return rate and the profit in dollars using Neural Net price predictions and precent price predictions.

5 Conclusion

We took advantage of the inherent volatility of Bitcoin prices to devise a high-frequency Bitcoin trading algorithm. Although our price prediction has only 60% accurate directionality, our action-selection algorithm has produced large profits over differing conditions, including those where the overall price trend is downward. We believe that the algorithm is good enough to power a Bitcoin trading bot for hobbyists.

Going forward, we plan to integrate our algorithm with available Bitcoin APIs to produce a functional Bitcoin trading bot. The main challenge will be to obtain data in real time.

References

[1] Shah, D., and Zhang, K. (2014). Bayesian regression and Bitcoin. arXiv preprint arXiv:1410.1231. http://arxiv-web3.library.cornell.edu/pdf/1410.1231v1.pdf

[2] Kim, K. J., and Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert systems with Applications, 19(2), 125-132.

[3] Kimoto, T., Asakawa, K., Yoda, M., and Takeoka, M. (1990, June). Stock market prediction system with modular neural networks. In Neural Networks, 1990., 1990 IJCNN International Joint Conference on (pp. 1-6). IEEE.

[4] Kamijo, K. I., and Tanigawa, T. (1990, June). Stock price pattern recognition-a recurrent neural network approach. In Neural Networks, 1990., 1990 IJCNN International Joint Conference on (pp. 215-221). IEEE.

[5] A. W. Lo and A. C. MacKinlay, Stock market prices do not follow random walks: Evidence from a simple specification test. Princeton, NJ: Princeton University Press, 1999.

[6] http://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter

[7] http://www.coindesk.com/price/#2013-12-16,2014-11-09,close,bpi,USD

[8] https://www.bitstamp.net/fee_schedule/

[9] https://blockchain.info

[10] https://www.okcoin.com