ESE 499 Capstone Design Project Proposal

Submitted to Professor Trobaugh and the Department of Electrical and Systems Engineering

Feedforward Machine Learning Systems to Improve Cryptocurrency Pricing Models

February 2017 - May 2017

Author: Sam Jones

Degree: B.S. in Systems Science & Engineering; Minors: Computer Science, Physics

Contact: samtjones@wustl.edu

Advisor: Professor Zachary Feinstein

Abstract

Cryptocurrency is a new asset class that struggles attracting investors with low risk tolerance due to their high volatility. The goal of this project is to use historical cryptocurrency trade data and machine learning models to generate prices that are better than market prices. This project implements an autoregressive linear model, several baseline machine learning models, and systems of machine learning models. It tests the accuracy of each in predicting the future 1 hour percent change in price of 10 different cryptocurrencies. Several models show significant predictive power and potential to make markets more efficient. However, this project is not able to conclude that margin trading cryptocurrencies provides a higher return on investment than a buy and hold strategy.

Background

Cryptocurrency is digital money that combines the most advanced cryptographic techniques with a revolutionary new platform called a blockchain to allow its users to securely transfer value between each other. The first ever cryptocurrency, Bitcoin, was created by a cryptographer under the pseudonym of Satoshi Nakamoto in 2009 and its value has since grown to a total market capitalization of roughly $20 billion at the beginning of 2017. Despite this spectacular rise in value, Bitcoin is far from perfect. Bitcoin transactions take many minutes to confirm, fees for transactions can often reach levels higher than its users are willing to pay, and its decentralized governance model makes it tough for developers to improve the underlying protocol. As a result of these issues, many new and separate cryptocurrencies have been created in effort further improve upon the Bitcoin software. In fact, there are now hundreds of cryptocurrencies that are valued at over $1 million in market capitalization, each with different software governing how they can be used. One particular cryptocurrency, Ethereum, has recently been threatening to take over Bitcoin as the cryptocurrency with the largest market cap, increasing by roughly 5x in value over the first quarter of 2017. Ethereum hopes to offer a turing-complete and programmable blockchain which can be used to create things like smart contracts and has many far reaching applications.

Although cryptocurrencies are an exciting new asset class that have the potential to revolutionize many industries, they have a major problem. Investors have not yet come up with a proven way to fundamentally value cryptocurrencies similarly to how they value stocks. This makes cryptocurrency extremely volatile. Its price can increase or decrease by orders of magnitude within a month or even a day. This is a major turnoff to new investors who may be interested in cryptocurrency but can not afford to risk losing half of their investment in a single day. The cryptocurrency industry loses out on much of the wealth that would come from these

new investors seeking a safe investment. More stable prices would help increase adoption of cryptocurrency greatly. Unfortunately, this is a very difficult task. Since prices are largely driven by speculation, the most practical way to value each cryptocurrency is to predict its future price based on past price data. One of the goals of this project is to come up with a model that uses past trade data to value cryptocurrencies better than market prices.

A common method used in systems engineering to estimate a future value in time series data is to treat the time series data as a stochastic process and use an *autoregressive linear* estimator. This model assumes that the output depends linearly on the previous values of the output and on a stochastic error term. For predicting cryptocurrency prices, the output I will be measuring is the future percent change in price over some time interval. The autoregressive linear model therefore takes the past changes in price over a set of different time intervals as input and estimates the future percent change in price using a linear regression.

However, there is much more information contained in historical trade data that could be useful in predicting the movement of cryptocurrency. A more complex model is needed in order to take advantage of this information. To build such a model, I will use machine learning. The most basic machine learning model is *multiple linear regression*. This method fits a linear relationship between many different independent variables and the dependent variable, the future percent change in price. Although it often works as a good baseline model, it assumes that the variables do not interact with each other, have a normal distribution, and can be transformed to fit a linear relationship with the dependent variable.

When a data set is still too complex for multiple linear regression, researchers often turn to a *random forest* to deal with the interactions between variables. A random forest is a collection of decision trees. Each decision tree iteratively buckets data into smaller subsets until each subset consists of similar samples. The decision tree then groups new data into these

subsets and each tree is pooled together by the forest to reduce bias and prevent overfitting. The critical parameters to tune in a random forest regression are the number of decision trees to use and the minimum size of the decision tree subsets. The issue with random forests is that they are not designed for continuous time series data and the input features need to be broken up into discrete variables to work with the model.

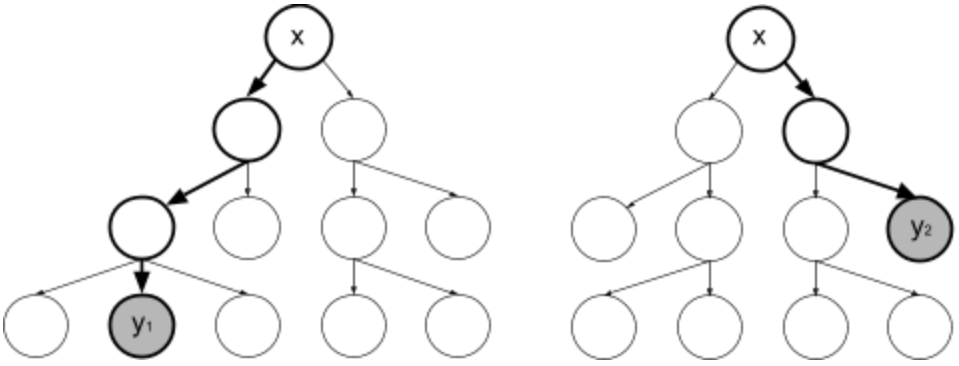


Figure 1. Random Forest of Decision Trees

To cope with the temporal nature of asset prices, researchers often use time series analysis methods instead. *Hidden markov models* are one of the most proven methods of time series analysis. They are designed to sequence together probabilistic transitions in order to estimate the final output state. The most important parameters in this model are the type of model (multinomial, gaussian, or gaussian mixture) and the number of hidden states. Although they work well with time series data that follow obvious trends, they cannot handle as much information as other machine learning models and often overfit on past trends.

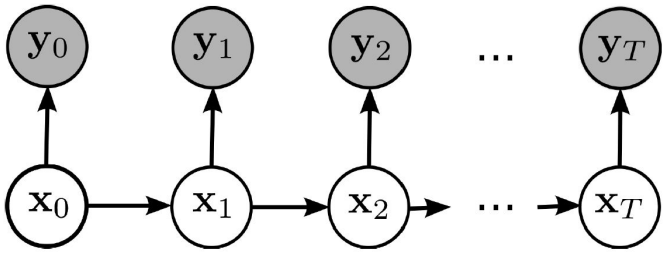


Figure 2. Hidden Markov Model

However, there is a time series model that can handle more information, called a *recurrent neural network*. Long short term memory networks are the most useful type of recurrent neural network for financial time series data. This model combines hidden layers of a stationary neural networks with an additional sequencing memory later to predict future data. The important parameters of a general neural network is the type of solver, the number of hidden neuron layers, and the number of neurons per layer. With recurrent neural networks, the number of past states to examine is an important parameter as well. Although recurrent neural networks are a more powerful than the other methods mentioned above, it requires the most data and it is very hard to tune the parameters without overfitting.

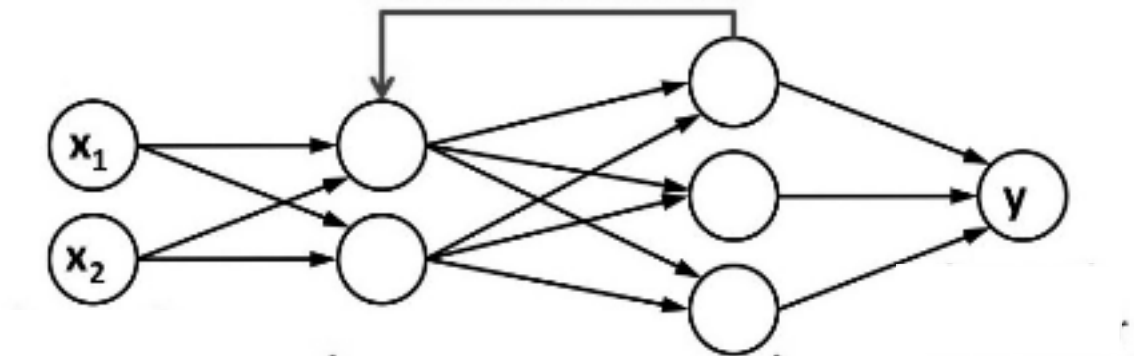


Figure 3. Recurrent Neural Network

Although each of these models can be very useful in modeling cryptocurrency prices, they all have weaknesses and need significant processing in order to recognize some of the many complex patterns that can exist in price data. The main goal of this project is to design a system of machine learning models that combines these models to capitalize on their strengths without being held back by their weaknesses. I will use historical pricing data to backtest the effectiveness of different models with the ultimate goal of implementing an algorithm that offers buy and sell prices that beat market valuations. If a machine learning model or system of them can be created that better values cryptocurrencies, its implementation would help reduce volatility and provide significant benefit to the marketplace.

Methods

Most aspects of this design project involved programming. In order to create fast and functional software, I used python. Although other languages such as C++ and Java can be used to build a more efficient trading algorithm, python is better for prototyping and has many open source libraries that are useful for building the models I will need in this project.

The first step of building machine learning models is collecting data. I gathered historical market trade data over the past year for 10 different cryptocurrencies and converted them into separate data tables. The process of how I collected data is covered in more detail in the following section. Once the data was collected, I filtered it into only the most significant trades, called “triggers.” Since the machine learning models look for patterns in past data, it is important to only feed the models significant and independent data points so that they do not overfit the data. For my project, I chose triggers to be only trades that had volume twice the size of the average trade over the past hour.

Once the data was filtered, I created the feature vectors that accompanied each trigger. Although the exact feature vector used in each machine learning model sometimes depended on the model, I first needed to create a baseline feature vector that could later be manipulated to create new features. In order to keep the focus on of this project on the machine learning models rather than the features, I decided to keep the feature vector simple. Each feature vector consisted of only three types of features: the past percent change in price, volume traded, and volatility. Each type of feature was calculated over 12 different time intervals ranging from 1 hour to 12 hours back for a total of 36 features. Each feature was calculated for every trigger in the data table. In addition to the features, I also calculated the dependent variable for each trigger. The dependent variable I chose to predict was the percent change in price over the next 1 hour period.

The next step of the design process was to split the data table into training and testing sets in order to train each machine learning model and backtest the accuracy of their predictions. To do this, I separated the data table of triggers and their corresponding features into the first 11 months for training and the last 1 month to testing. The train data set contained 2,074,526 triggers and the test data set contained 340,027 triggers. I also broke the train and test data tables into the feature vector and the prediction values. The resulting data tables from these splits were the input train set, output train set, input test set, and the output test set.

Once the data was filtered, I needed to implement the machine learning models and train them on the training set. For the baseline machine learning models, I used python scripts that others had created and open sourced online. The baseline model algorithms I implemented included all five of the models discussed in the introduction: autoregressive linear, linear regression, random forest, hidden markov, and recurrent neural network. For the autoregressive linear model and hidden markov model, the only features used were the past percent change in

price over the different time intervals. The other models used these features as well as the volatility and volume traded features. I discovered roughly optimal parameters for each model using a grid search. I trained each model using the input and output training sets, then used the input test set to create a vector of predicted percent change in prices.

After implementing the baseline models, I then focused on designing machine learning systems that would be improvements over the baseline model. I created four different machine learning systems that each used a combination of the baseline machine learning models and features from the original feature vector, as well as new features derived from the original features. The first model I created used the recurrent neural network to predict values of the three distinct features over the future hour interval and then used a random forest on only the future features to predict the percent change in price over the next hour. A basic diagram of this model is shown in Figure 4 below. In the diagram, x_i represents a specific feature and t_i represents a specific time interval.

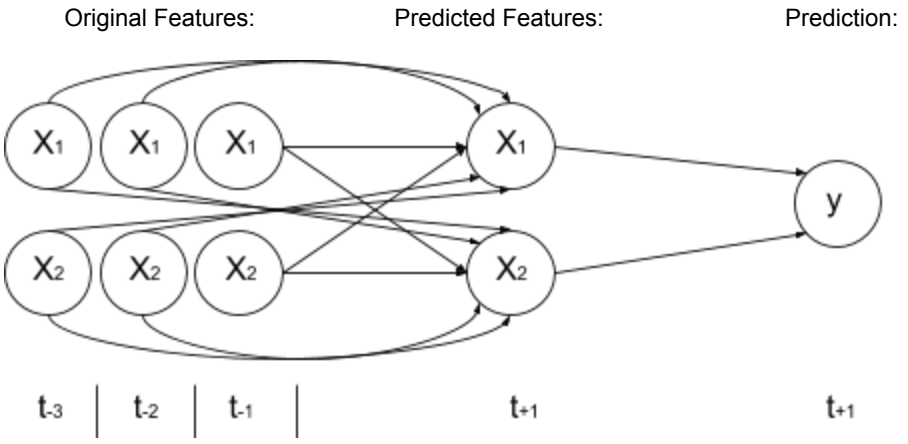


Figure 4. First Design Model

The second model I created was similar to the last model, except I used the recurrent neural network and hidden markov model to predict future features over 12 equally spaced time intervals ranging from 1 hour to 12 hours in the future. I again used a random forest on these new features to predict the percent change in price. This model is shown in Figure 5 below.

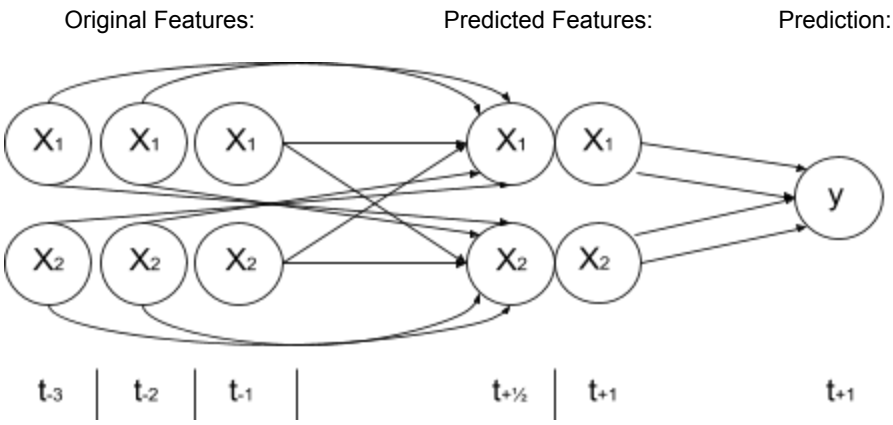


Figure 5. Second Design Model

The third model I created was also similar to the last model except I input the future features and the past features together into a random forest model to predict the percent change in price. Although a diagram of this model is too messy to depict, it is similar to the last diagram with the only difference being that each of the original features are also inputs to predict the final output variable.

The fourth model I created went in a different direction than the last three models. Instead of using predicted features as inputs to a final model, I ran all five baseline machine learning models to come up with five different values for the 1 hour predicted change in price. I then input these values as features in a random forest to combine the original predicted values into a single value. A diagram of this model is shown below.

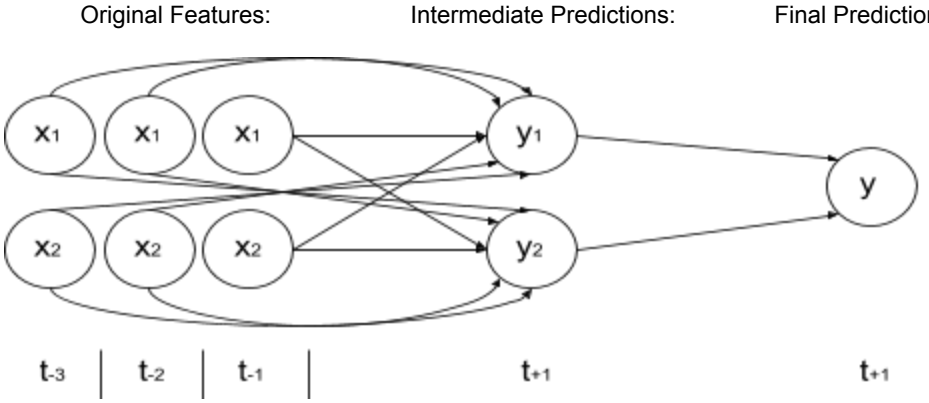


Figure 6. Fourth Design Model

Data Collection

To implement my machine learning model and compare it to other models, I needed cryptocurrency pricing data that was as detailed as possible. Fortunately, a few cryptocurrency exchanges provide complete market trade data via public api requests. The exchange I am using allows you to pull every trade for the past year. Each trade includes the cryptocurrency pair, time, amount, price, and whether it was a buy or sell. I downloaded all market trade data over the past year and converted it into a data table. A section of the data table that contains only 10 trades for a specific cryptocurrency pair shown below.

	amount	date	rate	total	type
0	101.60396452	2017-04-09 00:36:39	0.03700000	3.75934668	sell
1	0.13368915	2017-04-09 00:36:39	0.03700001	0.00494649	sell
2	18.65514001	2017-04-09 00:36:39	0.03700001	0.69024036	sell
3	2.70270124	2017-04-09 00:36:39	0.03700002	0.09999999	sell
4	16.49078368	2017-04-09 00:36:39	0.03700011	0.61016081	sell
5	7.58764929	2017-04-09 00:36:39	0.03700013	0.28074401	sell
6	0.13777434	2017-04-09 00:36:39	0.03704000	0.00510316	sell
7	1.78434177	2017-04-09 00:36:39	0.03704000	0.06609201	sell
8	0.20000000	2017-04-09 00:36:29	0.03705699	0.00741139	buy
9	8.09563863	2017-04-09 00:36:27	0.03705699	0.29999999	buy
10	0.04256857	2017-04-09 00:36:26	0.03704000	0.00157673	sell

Figure 7. Trade Data Table

After collecting all market trade data, I also filtered the data into triggers and features, as described in the previous section. Each row of the data table corresponds to a trigger trade while each column corresponds to either a calculated feature or the calculated output variable. An example of this data table is shown below including the future 1 hour change in price as well

as the past change in price, volume traded, and volatility features over intervals of 1 hour and 2 hours. The other features have been omitted in so that the data table would fit neatly in this report. The numbers in the column headers correspond to the number of seconds in the feature interval.

	future_change_3600	volatility_3600	volume_3600	past_change_3600	volatility_7200	volume_7200	past_change_7200
0	-0.014088	0.005286	51.711439	-0.008094	0.007226	56.056749	-0.006751
1	-0.010645	0.005295	54.782804	-0.008348	0.007263	59.128115	-0.007091
2	-0.010538	0.005532	55.373085	-0.008904	0.007185	59.757137	-0.007987
3	-0.009462	0.005428	55.455433	-0.009099	0.007179	59.809685	-0.008529
4	-0.006870	0.005428	56.968455	-0.009111	0.007179	61.322707	-0.008627
5	-0.002626	0.005428	59.448322	-0.009262	0.007179	63.752574	-0.008833
6	-0.002518	0.005497	60.800585	-0.009350	0.007179	65.185447	-0.008938
7	0.003358	0.005497	72.432255	-0.009418	0.007179	76.817117	-0.009018
8	0.028382	0.005497	78.844712	-0.009838	0.007179	83.229574	-0.009210
9	0.042814	0.005497	81.756545	-0.010413	0.007179	85.942620	-0.009443
10	0.051351	0.005497	84.791303	-0.010759	0.007179	88.977378	-0.009575

Figure 8: Feature Data Table

If I wanted to test my machine learning system on other data sets, I could also collect data on national currencies, commodities, or stocks. Although data is available for these assets, trade data that is as detailed and looks over the same time scale as my cryptocurrency data set is prohibitively expensive to purchase for this project. There are also other time series data sets such as electricity pricing data that do not behave similarly to traditional assets. Although these data sets could also be used to test my machine learning system, they are not traded the same and different features would have to be created to in order to implement useful models.

Results

The final models I tested included the autoregressive linear model, multiple linear regression, random forest regression, hidden markov model, and recurrent neural network, as well as the four design models mentioned above. After implementing the code for each model, I trained each model on the same train data set and used the same test data set to create prediction vectors whose values each corresponded to a trigger in the test set. For each model, I plotted the actual change in price vs. the predicted change in price to visualize their accuracy.

To quantitatively determine which models performed best, I used 3 metrics to measure how close each vector of predicted changes in price were to the actual vector of changes in price. The first comparison test I used was a R^2 test. This test indicates how much of the variability in the real values is explained by the predicted values. The value of a R^2 test ranges from 0 to 1, where 0 means there is no correlation between the actual and predicted values and 1 means they are perfectly correlated. The biggest caveat with a R^2 test is that it does not tell you anything about the bias of predictions. If the predicted and actual values are highly correlated but there is an extreme bias, the R^2 may be high but the model is still not adequate.

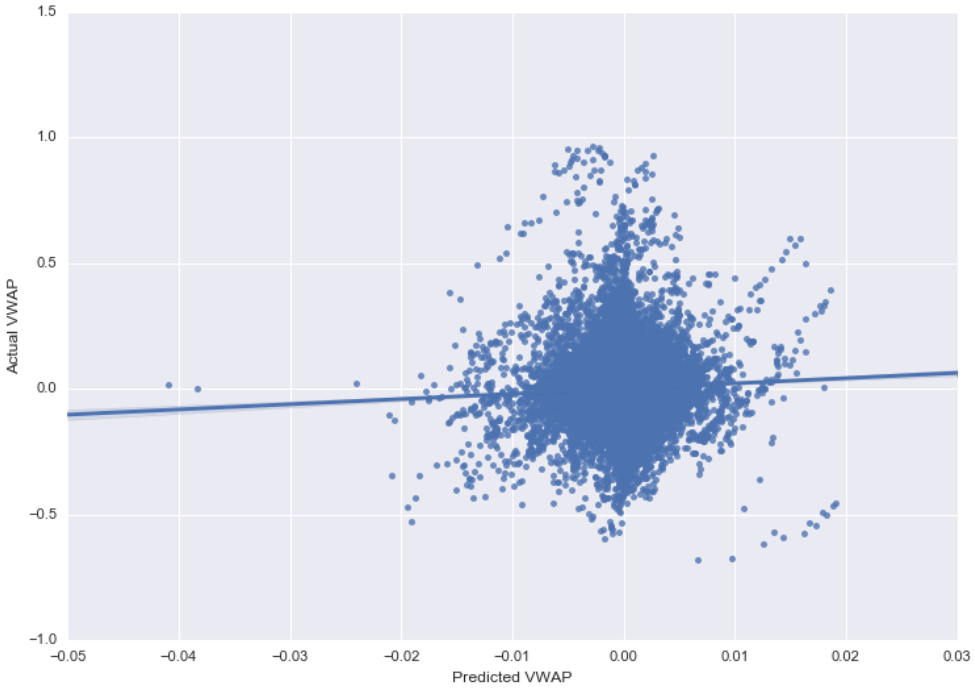
To get a better sense of the bias in each model, I measured the mean square error. This value tells you the average of the square of the error between the actual and predicted values. The higher the value, the greater the error, signifying the model predictions are less accurate. A model that makes predictions with high bias will have a high mean square error. However, the mean square error is not as useful a test as R^2 for measuring the variability.

The next parameter I used to measure the accuracy of models was the p-value of the correlation. This value tells you the probability of achieving such a correlation between the actual and predicted values if the data were random. Each p-value was extremely low due to the large sample size and did not provide useful information in determining which models were best.

Thus, this metric has been omitted. The scatter plots for each of the models as well as the R^2 and mean square error (MSE) metrics are shown below.

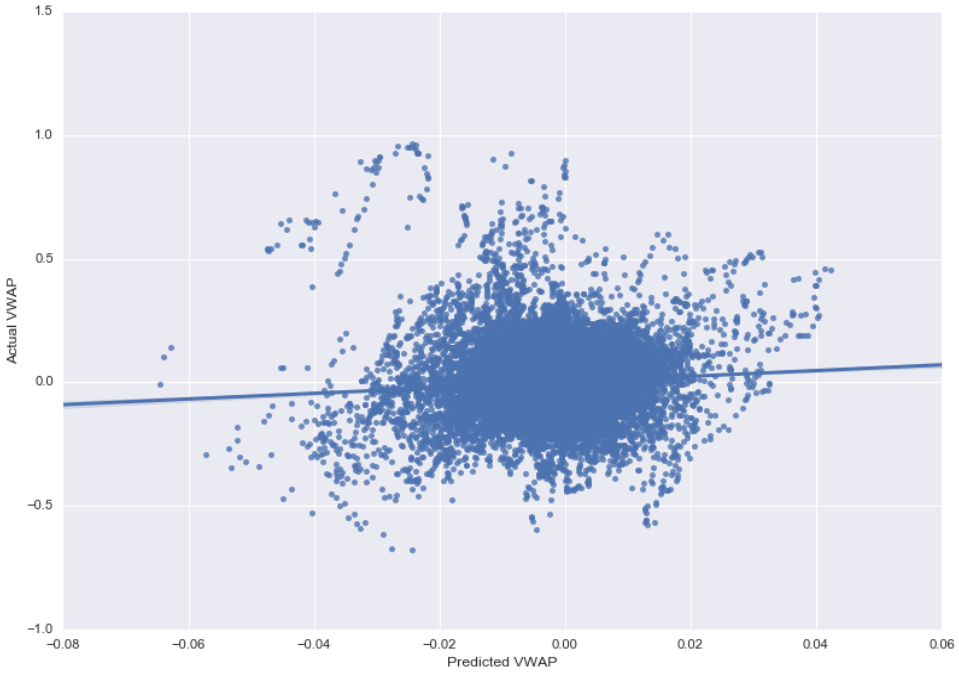
Autoregressive Linear

R^2 = 0.003039, MSE = 0.01495



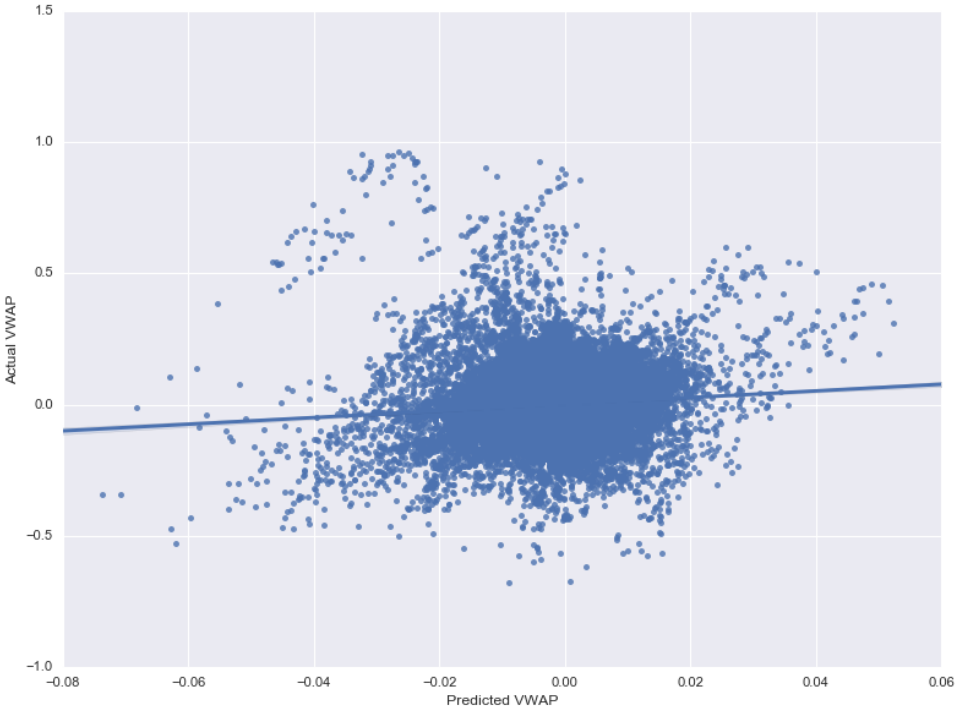
Linear Regression

R^2 = 0.005610, MSE = 0.01482



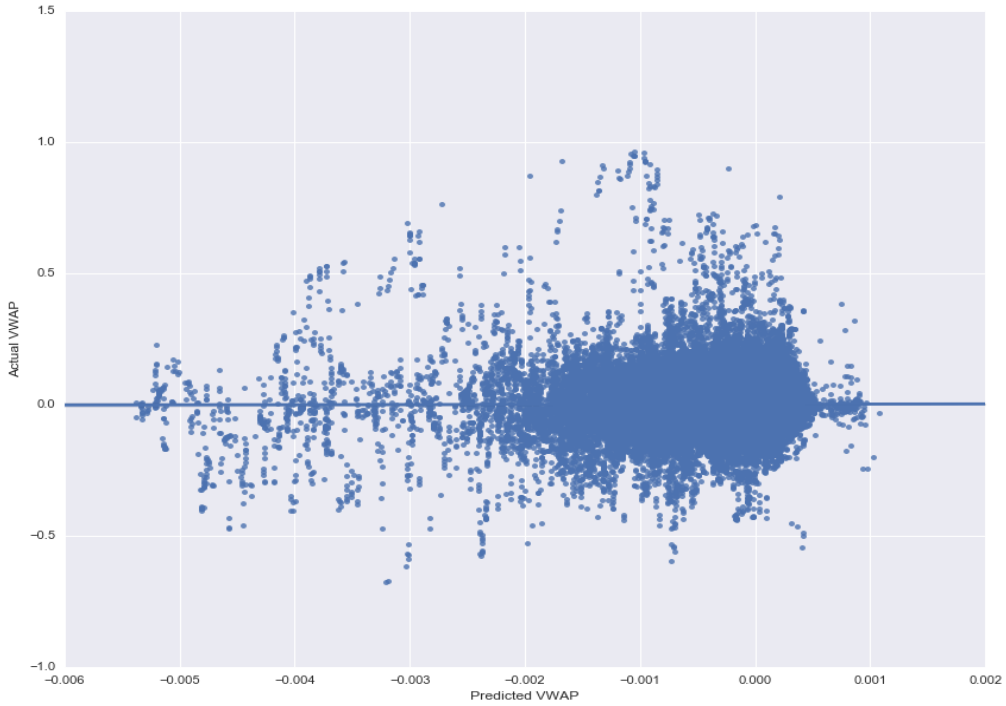
Random Forest

R^2 = 0.006939, MSE = 0.01479



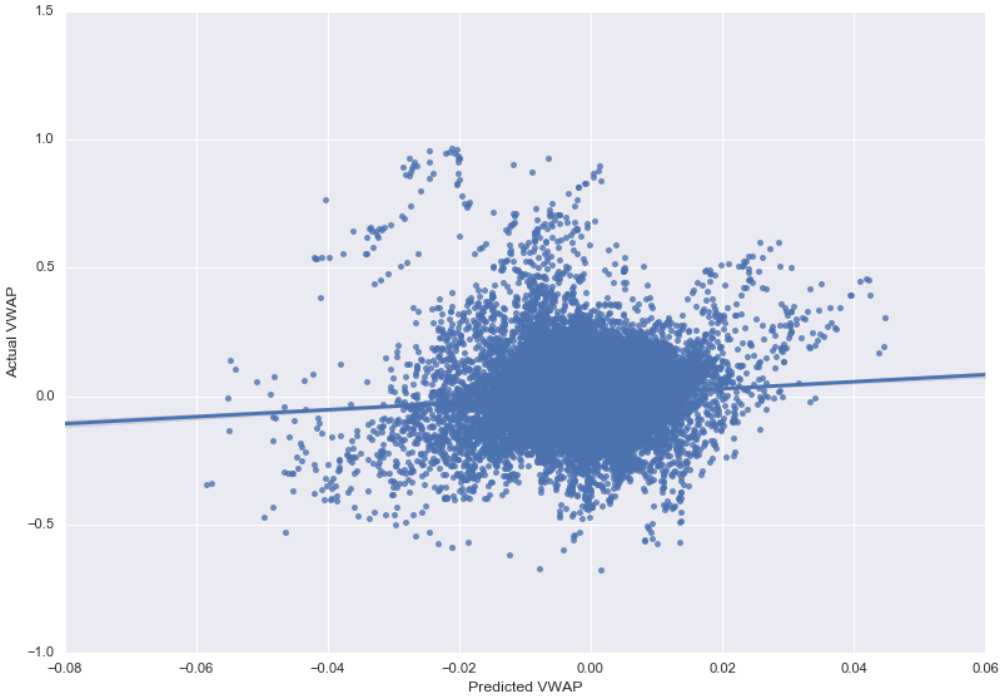
Hidden Markov

R^2 = 0.001296 MSE = 0.01498



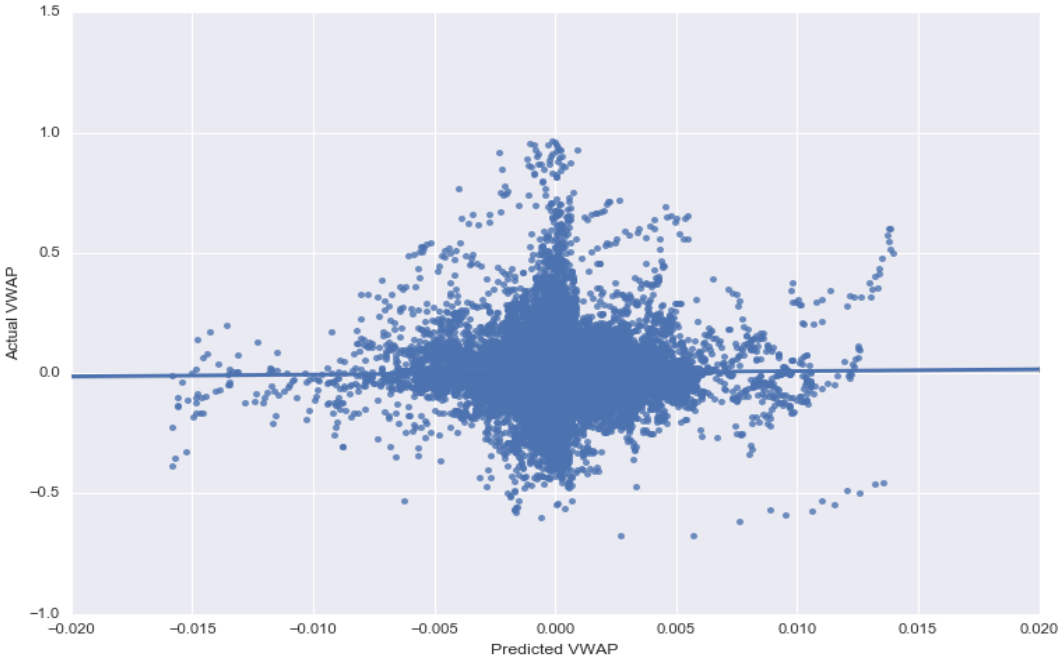
Recurrent Neural Network

R^2 = 0.007261, MSE = 0.01480



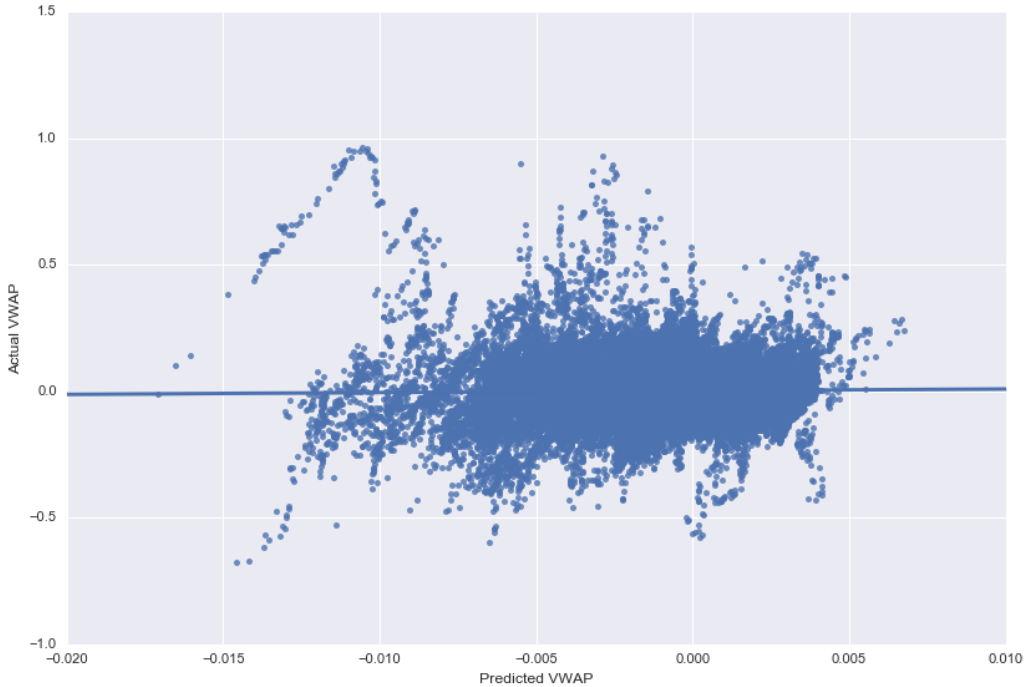
Model #1

R^2 = 0.000158, MSE = 0.001499



Model #2

R^2 = 0.000445, MSE = 0.01497

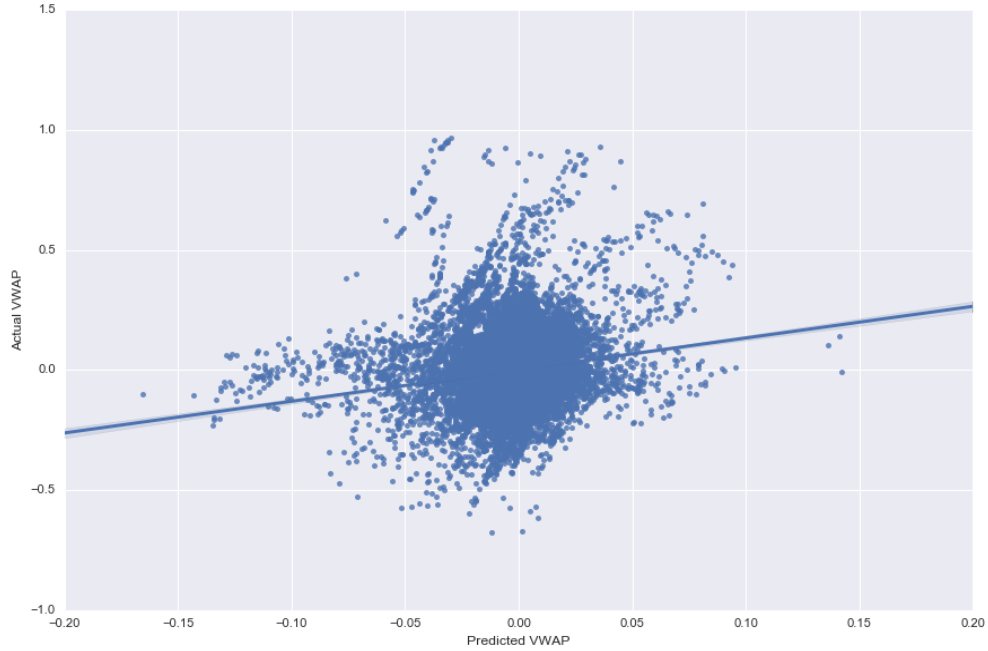


Model #3

R^2 = 0.008705, MSE = 0.01452

Model #4

R^2 = 0.01585, MSE = 0.01471



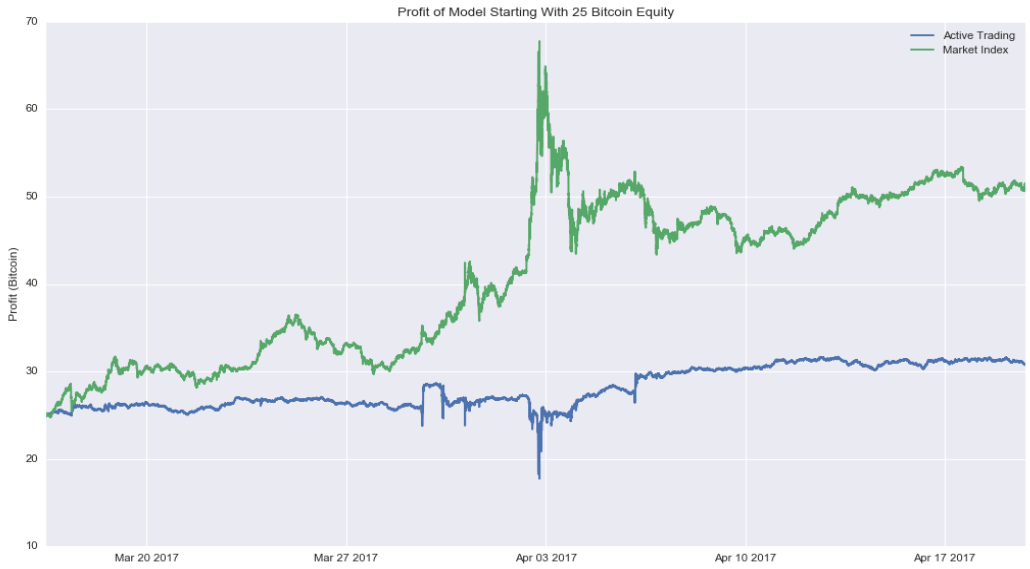
In addition to these statistical tests, I also created an algorithm that calculates the theoretical profitability of each of the models had it been trading over the testing period. For each trigger, the algorithm calculates a buy or sell amount proportional to the predicted change in price and places limit buy or sell order at the top of the orderbook. Since the exchange offers margin trading, the algorithm is allowed a net negative position in any cryptocurrency. The profitability algorithm accumulates all the buy and sell orders that were traded against before the next trigger and calculates the profit or loss of the trade over the next hour. Since the exchange charges a 0.10% fee for each limit order, 0.20% of each trade was deducted from the total profit for trading into and out of that position. The net profit of every trade was recorded in a new data table and graphed over the testing periods for each model. I also compared each model to a buy and hold strategy where each of the 10 cryptocurrencies were bought in equal amounts at the beginning of the testing period then sold at the end with no trading done in between. The profitability of the top performing models are shown below.

Random Forest

Profit = 5.980 Bitcoin, Return = 23.9%

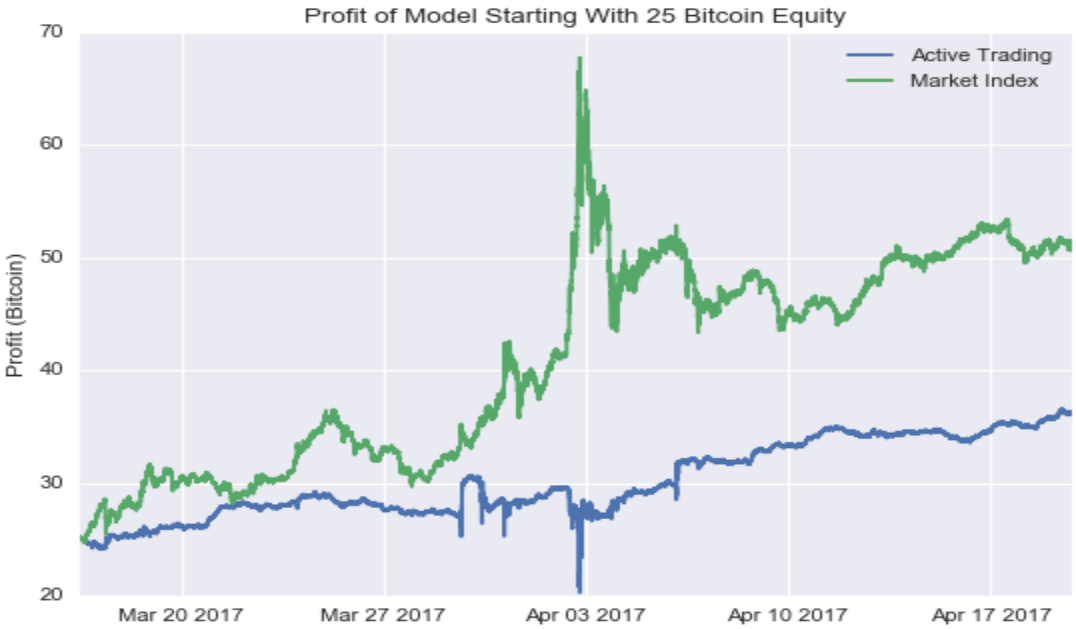
Model #3

Profit = 6.103 Bitcoin, Return = 24.4%



Model #4

Profit = 11.378 Bitcoin, Return = 45.5%



Discussion

Although none of the machine learning models were able to predict the future with extremely high accuracy, many of the models had significant predictive power and also showed improvement over the baseline models. The best model was the fourth model I designed that combined the predictive power of the baseline models. This model likely performed best because it weighted the predictions of multiple models to form a more complete understanding of the interactions between features. The model achieved the highest R^2 with a value of 0.01585, meaning that 1.585% of the variability in the future percent change in price was predictable by the model. The model also had a low mean squared error with a value of 0.01471, meaning that the average of the square of the difference between what was predicted and the actual value was 1.471%. Also, using the simple trading strategy detailed above, the model achieved a fairly low risk profit of 11.378 bitcoins on an original 25 bitcoin investment over the month period, which equals a return on investment of 45.5%.

The second best model was the third model I designed that combined future features with past features in a secondary model. The secondary model used that had the best results was the random forest. This model likely performed well because it contained the most feature information in the secondary model. It achieved a R^2 of 0.008705 and a had the lowest mean square error of 0.01452. This model also achieved a low risk profit of 6.103 bitcoins on an original 25 bitcoin investment for a return of 24.4% that month. Although both models performed well, neither were able to beat the market index over the testing period. The buy and hold strategy over the month period achieved a profit of 26.550 bitcoins for a total of 106.2% return on investment. This is an extremely high return, making it very difficult for any model to beat it over the testing period. Still, both of these models showed improvement over the baseline models, as well as the first two design models I created.

Although it is important to test any trading model over periods of both upward and downward market movements, I was not able to do so over an entire month period. For the past several months, the cryptocurrency market index has consistently risen. A graph of the total market capitalization of all cryptocurrencies is shown in Figure 9 below. The market cap increased from roughly $11 billion on September 1st, 2016 to $36 billion on April 30th, 2017. Since I only had 1 year of trade data, every testing period I could have used either experienced an increase in the market index or did not have enough data before that period to effectively train the models. Even though a full month of negative price movement was not tested on, there were still periods of several days that experienced a significant downturn. An example of a downturn can be seen between April 3rd and April 12th in the profitability graphs above. Fortunately, each tested model actually performed better over this period than over any other part of the month. This is likely because the trading algorithm can 'short' cryptocurrencies, effectively having a negative position, to make money whenever the market becomes temporarily overpriced. This is important because the significant positive returns on the market index are likely not sustainable forever and there will eventually be longer periods of downward trends in the market.

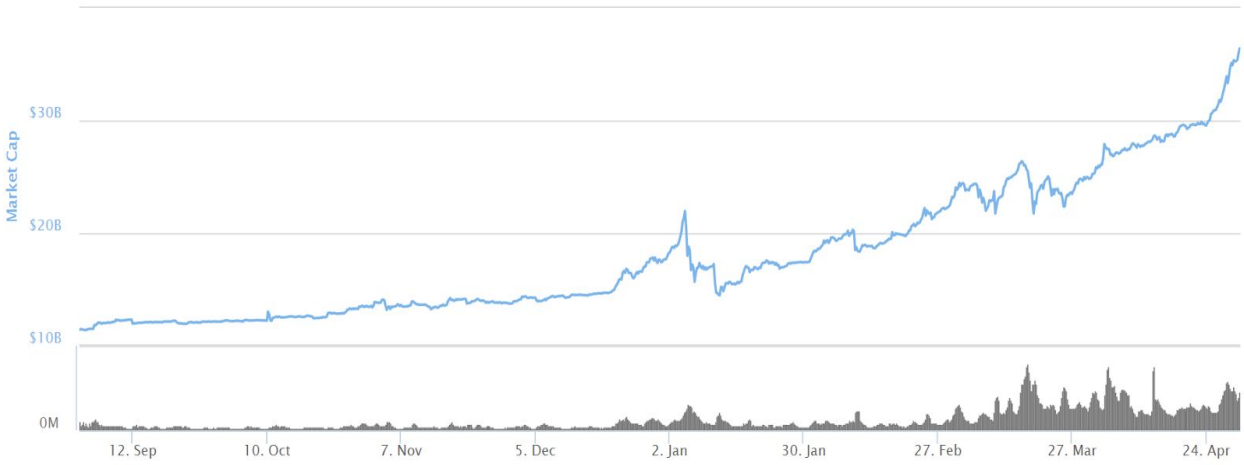


Figure 9. Cryptocurrency Market Index

The first two models I designed did not perform nearly as well as the last two. The first model I designed only achieved a R^2 of 0.000158 and a mean square error of 0.01499, while the second model achieved a R^2 of 0.000445 and a mean square error of 0.01497. These models likely did not perform well because feeding the features forward resulted in a significant loss of information in the final models. The second model performed better than the first probably because there were more features and thus not as significant a loss of information in the final model. Neither were able to beat the best baseline models though. The best baseline models were the random forest and recurrent neural network. The random forest achieved a R^2 of 0.006939 and a mean square error of 0.01479. The recurrent neural network achieved a R^2 of 0.007261 and a mean square error of 0.01480. The worst model was the hidden markov model, which only achieved a R^2 of 0.0001296 and had a mean square error of 0.01498.

The accuracy metrics mentioned above were the most important considerations I had in deciding which models were best able to predict the future. However, in real trading, speed is also a very important factor in deciding which model to use. It is important that models are able to calculate a prediction before this markets move significantly. Fortunately, since each model was implemented from highly optimized third party code, each model was fast enough to calculate predictions on the entire test set within a few minutes. Since the test set included 340,027 triggers, a single prediction would only take a few milliseconds. This is more than fast enough to be used in real cryptocurrency markets. In fact, the limiting factors in performance was calculating the features and can take orders of magnitude longer than the predictions.

Although it would be nice to be able to compare the performance of my models to that of the past results of other researchers, there is no literature specifically on using machine learning to predict cryptocurrency prices. It is thus impossible to know if my results were within reasonable expectation given the features, models, and cryptocurrencies used. The baseline

models I implemented are useful for comparing to models commonly used in research given the same market environment and data. However, it is not safe to make assumptions about how the models I designed would perform on different data sets and feature vectors. Much more research would have to be done on different data to make conclusions on general time series data.

There are many different approaches that could be taken in the future to expand on what was done in this project and come up with more conclusive results. Testing each of the models over test sets other than the month I used is one way to confirm or reject the results found in this project. Selecting features other than the three distinct features over the specified time intervals is another way to come up with more results to make more distinct conclusions. Using completely different data sets for testing such as commodity or electricity prices would also be useful to determine if the results are dependent on using cryptocurrencies markets. There are also prediction variables other than the 1 hour percent change in price that each model could be used to predict, such as a 1 day percent change in price or a 1 hour percent change in volatility. Each of these techniques could offer great insight into why I obtained the results I did and show if my results are the standard or the exception.

Conclusion

Despite many of the models proving they were able to predict cryptocurrency movement significantly better than the market, none of the models performed better that the market index over the month test period. This is partially due to trading fees of 0.10% and the simplicity of the trading strategy. Still, this implies that a simple buy and hold strategy would be the best strategy to maximize return on cryptocurrency investment. This is partially due to extreme growth experienced by the cryptocurrency industry over the past few months. More testing, as discussed in the previous section, would need to be done over a more extensive period of time in order come up with more conclusive results. Although, at least during past several months, being exposed to the cryptocurrency market in general would have been a very rewarding investment.

References

1. Landen, Camilla. "Bond Pricing in a Hidden Markov Model of the Short Rate." *Finance and Stochastics* 4.4 (2000): 371-89.
2. Gupta, Aditya, and Bhuwan Dhingra. "Stock Market Prediction Using Hidden Markov Models."*2012 Students Conference on Engineering and Systems* (2012).
3. Nystrup, Peter, Henrik Madsen, and Erik Lindstrøm. "Long Memory of Financial Time Series and Hidden Markov Models with Time-Varying Parameters." *Journal of Forecasting* (2016).
4. "Exploring Permanent and Transitory Components of Stock Return." *Hidden Markov Models Advanced Studies in Theoretical and Applied Econometrics* (2009): 117-26.
5. Liatsis, Panos, Abir Hussain, and Efstathios Milonidis. "Artificial Higher Order Pipeline Recurrent Neural Networks for Financial Time Series Prediction." *Artificial Higher Order Neural Networks for Economics and Business* (2006).
6. Kamijo, K., and T. Tanigawa. "Stock Price Pattern Recognition-a Recurrent Neural Network Approach." *1990 IJCNN International Joint Conference on Neural Networks* (1990).
7. Sun, Xiang, and Yong Ni. "Recurrent Neural Network with Kernel Feature Extraction for Stock Prices Forecasting." *2006 International Conference on Computational Intelligence and Security* (2006).
8. Xie, Xin-Kai, and Hong Wang. "Recurrent Neural Network for Forecasting Stock Market Trend." *Computer Science, Technology and Application* (2016).
9. Zhang, Yingjian. “Prediction of Financial Time Series with Hidden Markov Models.” *Simon Fraser University* (2004).