Bitcoin Trading via Machine Learning Models

Jihan Yin
Department of Electrical Engineering and Computer Science, University of California, Berkeley
Berkeley, California 94704
jihan_yin@berkeley.edu

***Abstract* - This project revolves around the application of machine learning algorithms towards prediction of Bitcoin prices. We will be using two models, gradient boost and neural nets on a dataset of 14 features to try to predict daily price changes, through two methods - binary classification of changes in price, and regression on the ratio of change in price. Both gradient boost and neural nets were not successful in binary classifying price changes. They would only predict a positive change in price, and both had a 60% accuracy because 60% of our samples had a positive change in price. For regression, they did not do much better, almost always predicting positive ratio changes in price.**

***Keywords* - Machine Learning, Bitcoin, Neural Nets, Gradient Boosting, Stock Market, Cryptocurrency**

I. INTRODUCTION

Bitcoin is a is a cryptocurrency introduced in 3 January 2009[1]. It is a completely decentralized digital currency with transactions operating through peer-to-peer networks. All of the transactions are stored on an immutable public ledger, known as a blockchain. The nature of a blockchain makes it such that altering data on the blockchain is incredibly hard, making bitcoin secure. Because of the anonymous and secure nature of Bitcoin, its popularity has been increasingly steadily since its formation. In the past year, more established firms and banks have started to show their interest in bitcoin, leading to a rise in value much steeper than that of previous years. However, for bitcoin to rise past its current niche, it must become more accepted by legitimate businesses as currency and gain more widespread use through regular business transactions.

Bitcoin can be traded the same way as stocks on the stock market. There are many websites acting as bitcoin asset brokers, but for this project, we will be focusing on trading through website Coinbase. Ever since its inception, machine learning models have been successfully made to predict changes and trends in the stock market due to relationships in price with measurable real world quantities. This is the motivation behind this project, as we will be pulling data off the blockchain as features for predicting Bitcoin trends.

II. DATA COLLECTION

All of the data and code used for this project can be found in the github source[2]. The data was pulled from the bitcoin blockchain[3], where we selected 13 features out of intuition in their relationship with Bitcoin value, shown in table 1. Data off the blockchain website only gives daily bitcoin value, although other features may be given in shorter intervals. Because we are trying to predict daily value changes, we will only be taking the daily values for each feature. If there are multiple values per day, we will take the value corresponding to the latest time in that day.

Feature	**Description**
Total Bitcoins	Total Bitcoins mined
Market Price	Average Bitcoin market price in USD
Market Cap	The total USD value of bitcoin supply in circulation
Trade Volume	The total USD value of trading volume of Bitcoin
Hash Rate	Estimated tera hashes per second of the Bitcoin network
Cost Per Transaction Percent	Miners' revenue as percentage of transaction volume.
Unique Addresses	Unique addresses on the Bitcoin blockchain
N-Transactions	Number of daily Bitcoin transactions
Estimated Transaction Volume - USD	Estimated transaction value in USD
Estimated Transaction Volume	Total estimated transaction value on the blockchain
Output Volume	Total value of all transaction outputs per day
UTXO Count	Number of unspent Bitcoin transaction outputs
N-Transactions Excluding Popular	Number of daily Bitcoin transactions outside of the 100 most popular trading sites.

Table 1 - List of features and descriptions based off the blockchain[3].

We will be splitting the data in two parts for training and validation, and scaling them to unit length.. We will be training on the first 70% of data, and validating on the last 30% of data, chronologically. The reasoning behind this is that we expect an underlying relationship behind our features and Bitcoin price changes unaffected by time, and so our model trained on the training data should perform just as well on the validation dataset.

III. MODEL PREDICTION

We will be using two types of models - classification and regression. For classification, we will be classifying whether the price will rise (label 1), or drop (label -1). For regression, we will be trying to predict the ratio of price change - Price(day + 1) / Price(day). We will attempt to use a neural net and gradient boosting for each of the two types, using grid search to find the best parameters for each model.

We will be using Sklearn's Multi Layered Perceptron for neural net classification and regression, as well as Sklearn's Gradient Boosting class for gradient boost classification and regression. For the neural net, we will be grid searching through what activation function to use (relu units or tanh), the hidden layer sizes as well as the neurons in each layer, and alpha. For gradient boost, we will be grid searching through the number of estimators, the learning rate, the max depth, and the minimum samples per leaf. Our default parameters for the neural net is the defaults of Sklearn with the random_state set to 189. Our default parameters for gradient boosting is 30 minimum samples per split, using 'sqrt' for max_features, with subsample being 0.8 and random_state set to 189[4]. The following graphs show the accuracy of different values for each parameter for our neural nets with all other parameters having their optimized value through grid search.

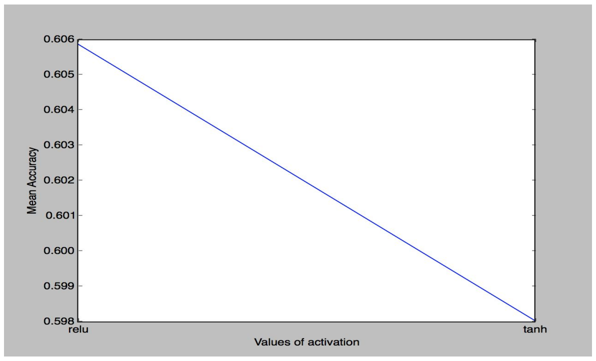

Fig. 1 - MLP Classifier Activation Values

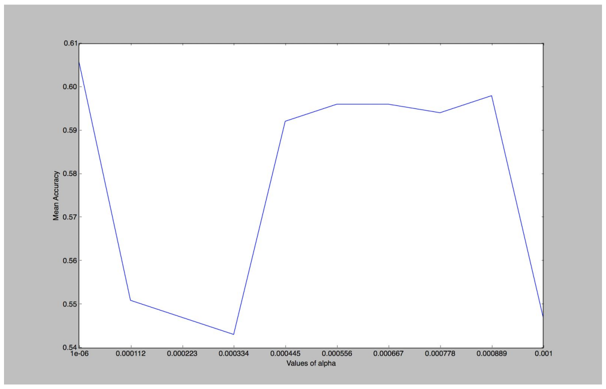

Fig. 2 - MLP Classifier Alpha Values

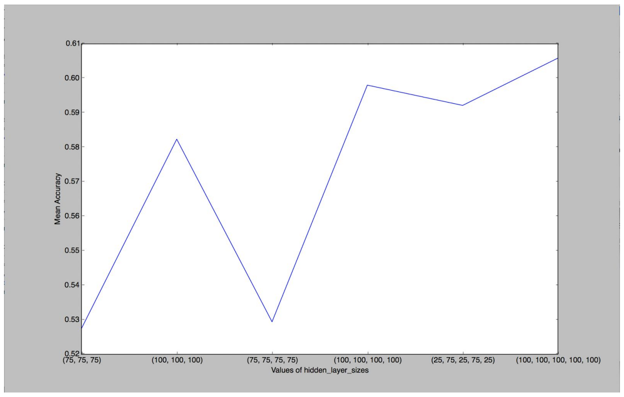

Fig.3I - MLP Classifier Hidden Layer Values

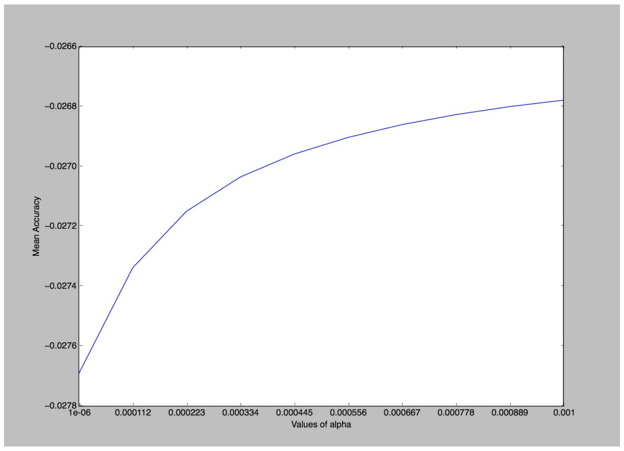

Fig. 4 - MLP Regressor Alpha Values

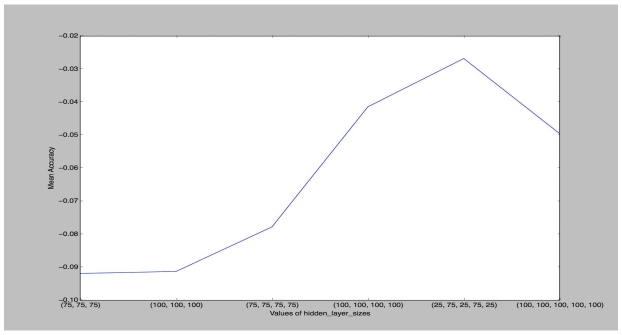

Fig. 5 - MLP Regressor Hidden Layer Values

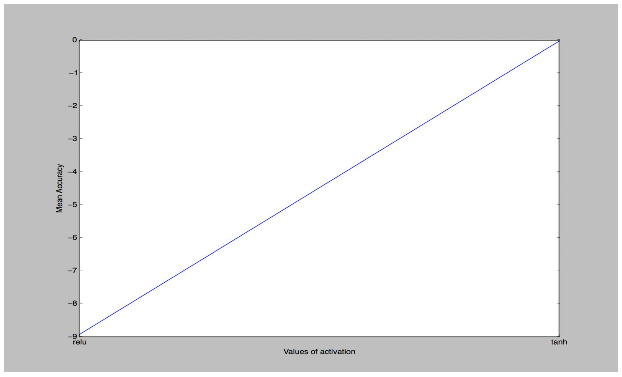

Fig. 6 - MLP Regressor Activation Values

Through grid search, we found that the best parameters for our neural net classifier was an alpha value of 1e-6, using relu units and 5 fully connected layers of 100 neurons each. For the neural net regressor, we found that the best parameters was an alpha value of 0.001, using tanh as its activation function and with layers of (25, 75, 25, 75, 25) neurons. This was very interesting as we expected relu to function better for regression, and tanh for classification. We did not achieve a very high accuracy for classification. In fact, upon a closer look in our neural net classification, our model classifies every single samples as a '+1', indicating an increase in price. We get an accuracy of ~60% purely because 60% of the samples have an increase in price. Fig. 4 shows a steady pattern relating our alpha values to accuracy, with higher alpha values leading to better accuracy. For classification, however, fig. 2 is very noisy and no clear analysis can be made. fig. 3 and 5 show a relationship of more hidden layers leading to higher accuracy, although for regression, our specified layers of (25, 75, 25, 75, 25) neurons did better than 5 layers of 100 neurons each.

The following graphs show the accuracy of different values for each parameter for our gradient boost with all other parameters having their optimized value through grid search.

From our graphs, we can see that the minimum samples per leaf, max_depth, and n_estimators do not matter for our gradient boost classifier, with accuracy barely changing for different values of learning rate. Again, looking into our classification, we classify every sample as a '+1', resulting in an accuracy of 60%. For regression, however, it appears that gradient boost does better than neural nets. We see that as we use more estimators, our accuracy goes down, and as learning

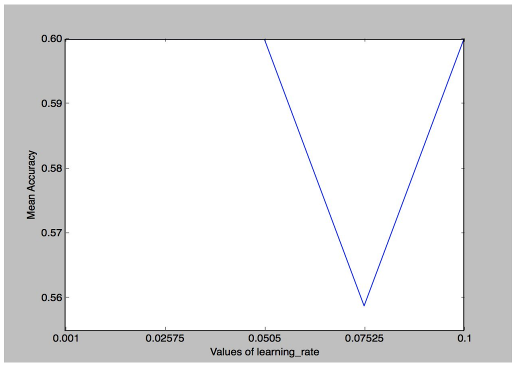

Fig. 7 - GB Classifier Learning Rates

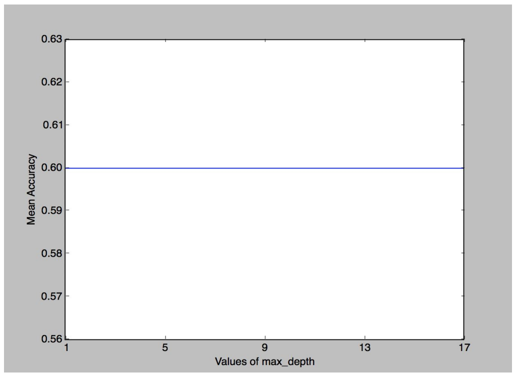

Fig. 8 - GB Classifier Max_Depth

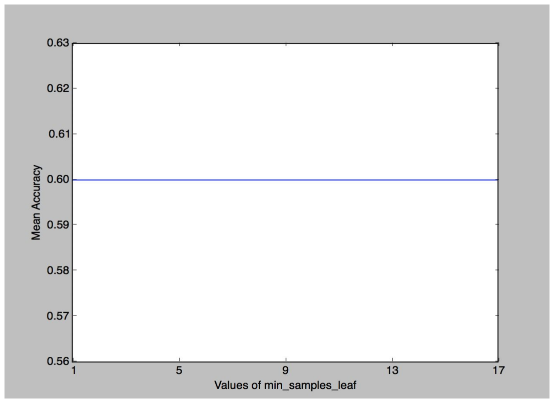

Fig. 9 - GB Classifier Min Samples

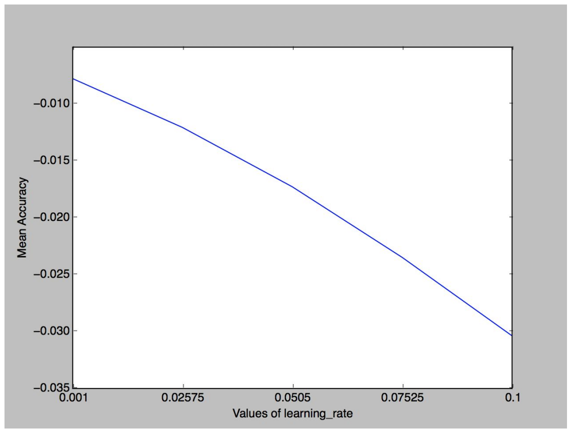

Fig.12 - GB Regressor learning_rate

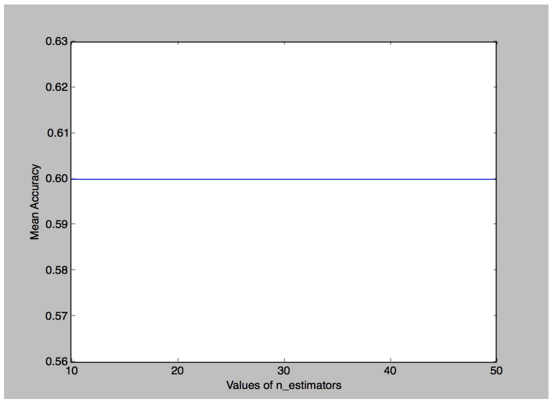

Fig. 10 -GB Classifier n_estimators

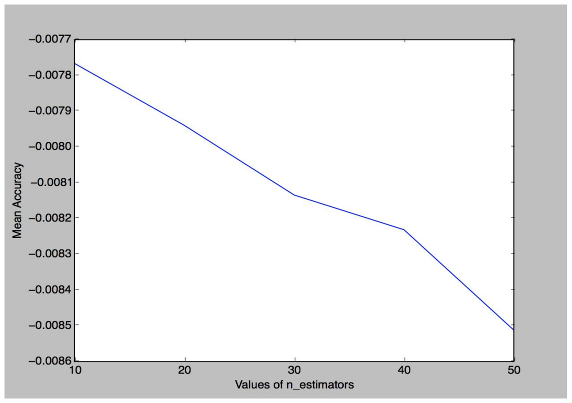

Fig. 13 - GB Regressor n_estimators

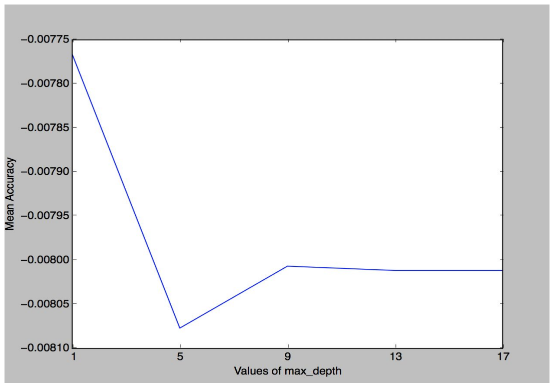

Fig. 11 - GB Regressor max_depth

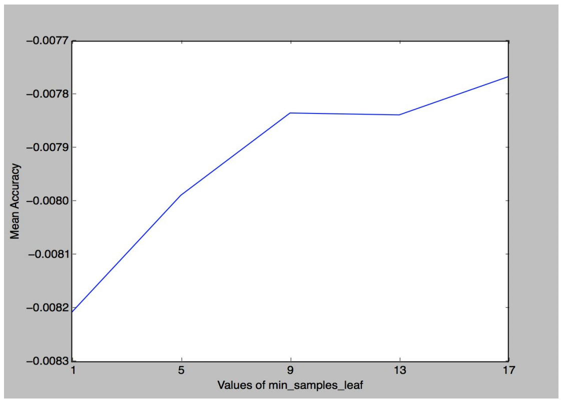

Fig. 14 - GB Regressor min_samples_leaf

rate increases, accuracy decreases. Fig. 11 shows that after a certain value, max_depth has no effect on our model. Fig. 14 shows that increasing the minimum samples per leaf increases our accuracy, which makes sense due to forcing our model to make better separation thresholds. Overall, our neural net and gradient boost models both are not accurate in classifying rises or drops in price, but seem to do okay in trying to predict the ratio change in price.

IV. Validation Simulation

To compare our final models against each other, we will be simulating actively using our model to invest in Bitcoin on the validation dataset through Coinbase, which charges a 1.5% fee on buying and selling[5]. We start with $10,000 in capital, and use the following algorithm for investing:

```
Algorithm 1 Regression Simulation
Input: Predicted Change, p = [0, inf)
Output: Percent to Invest
1: $Percent\_invest = max(0, 1 - e^{-2(p-0.8)})$
2: return $Percent\_invest$
```

```
Algorithm 2 Classification Simulation
Input: Predicted Change, p = {-1, 1}
Output: Percent to Invest
1: if ($p = 1$) then
2:     $Percent\_invest = 0.9$
3: else
4:     $Percent\_invest = 0$
5: end if
6: return $Percent\_invest$
```

For classification, we have a simple algorithm. For each day, if we predict that the price will go up, we invest everything we have. If we predict the opposite, we sell all our investments. For regression, we use the formula given to determine what percent to invest with the logic that we are more confident in our predictions of price increases than our predictions of price decreases, and so we invest exponentially more if we believe the price will go up, and we still invest a little bit if we believe the price will go down. We will be running all four optimized models on this simulation to see how well they will do, and compare them against the naive approach of investing everything in the beginning and leaving it there forever. Our results are the following graphs:

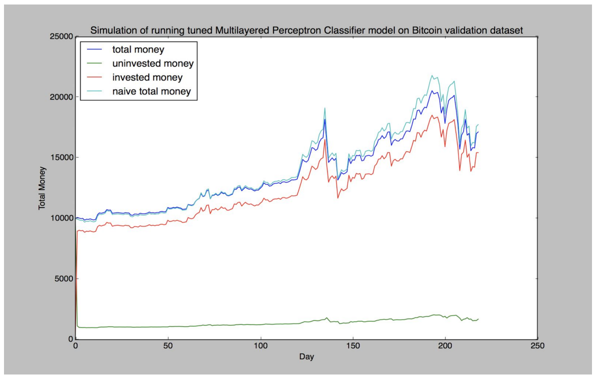

Fig. 15 - Performance of MLPC

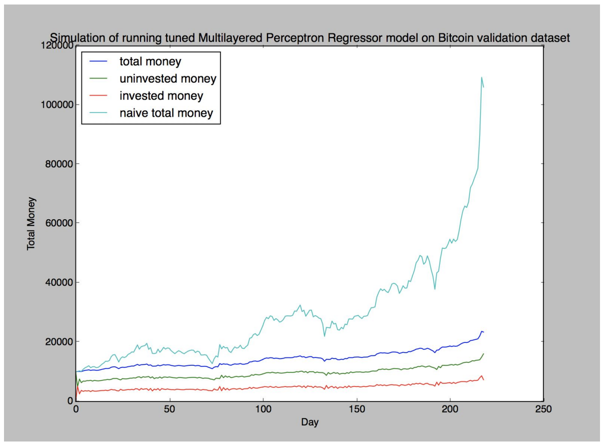

Fig. 16 - Performance of MLPR

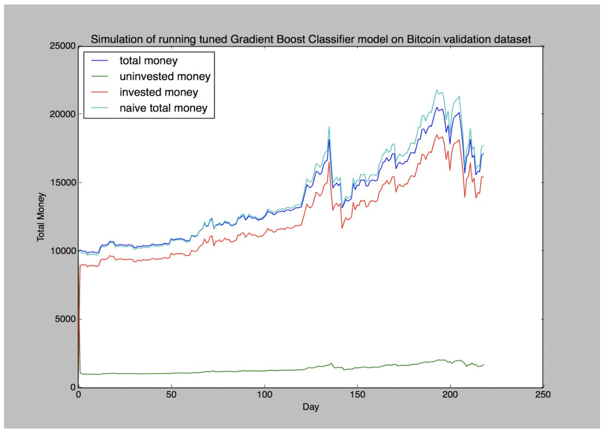

Fig. 17 - Performance of Gradient Boost Classifier

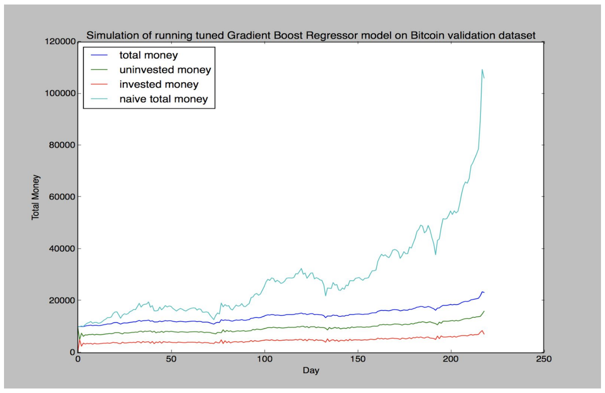

Fig. 18 - Performance of Gradient Boost Regressor

As we can clearly see from the graphs, our classifiers do much better than our regressors. However, this is because our classifiers always predict a price increase, and as such we are always investing 90% of our total capital. This is shown in figures 15 and 17, as the invested graph is just a shifted version of the naive graph, which invests all of our capital. Our gradient boost regressor ends with a total sum of $23,333, while our neural net regressor ends with a total sum of $23,447. These two models are extremely close to each other, and upon further inspection, they almost always predict a ratio of change greater than 1, doing slightly better than the classifiers at prediction.

V. CONCLUSION

Our four total models each were not successful in predicting the trends for bitcoin prices. Both of our classifiers classified everything as a positive change, and both our regressors predicted almost every sample to have a positive change. The reason our validation simulations did not result in a net loss is due to the nature of bitcoin in the past year, as it has consistently risen at a faster and faster rate. This can be seen in the naive approach doing better than any of our models. There are a few reasons as to why our models failed. It is possible that neural nets and gradient boosting are simply the wrong models to work with bitcoin, and there is some underlying relationship that could not be discovered with neural nets and gradient boosting. Since we only have 2 years' worth of data to work with, and we only train on 70% of the data, we only have 511 dimension-14 samples to work with. There may be too little samples to work with for our neural net, resulting in it having high variance and low bias, as seen in its inaccuracy. Gradient boost would also have too little samples to work with and too many dimensions to split on, resulting in inaccuracies that were shown in our data. The daily change of bitcoin prices may also be correlated with data from the past few days, whereas in this project, we only worked with the data of a single day. This may be another reason for our inaccuracy, since for each sample, we don't have all the features we need for prediction. To fix this, we would have to introduce many more features for each previous day that we care about. This would lead to much higher dimension data, and we will need more samples to work with to avoid the high variance resulting from such high dimensional data. We could write a program to mine data directly off the blockchain instead of taking our data from Blockchain's public database, or just take snapshots of our features with a frequency higher than once per day. With more accurate data and better models, we can find more success in predicting bitcoin price changes.

REFERENCES

[1] - *Bitcoin*, Bitcoin Project 2009-2017, web, accessed 27 November 2017, https://bitcoin .org/en/faq

[2] - Arilato, *Bitcoin Predict*, Github Inc. 2017, web, Accessed 9 December 2017, https://github.com /arilato/BitcoinPredict

[3] - *Blockchain*, Blockchain Luxembourg S.A. 2017, web, Accessed 27 November 2017, https://blockchain.info

[4] - Aarshay Jain, *Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python*, Analytics Vidhya, Analytics Vidhya 2013-2017, web, Accessed 29 November 2017, https://www.analyticsvidhya.com/ blog/2016/02/complete-guide-parameter -tuning-gradient-boosting-gbm-python/

[5] - *Coinbase Pricing & Fees Disclosures*, Coinbase 2017, web, Accessed 3 December 2017, https://support.coinbase .com/customer/portal/articles/2109597