Spring 6-1-2022

# Machine Learning and the Network Analysis of Ethereum Trading Data

Santosh Sivakumar
Santosh.Sivakumar.22@Dartmouth.edu

# Machine Learning and the Network Analysis of Ethereum Trading Data

Santosh Sivakumar

Undergraduate Computer Science Thesis

*Advised by*

Professor Dan Rockmore

Dartmouth College

Hanover, New Hampshire

June 1, 2022

# ABSTRACT

Since their conception, cryptocurrencies have captured the public interest, motivating a growing body of research aimed at exploring blockchain-based transactions. This said, little work has been done to draw conclusions from transaction patterns, particularly in the realm of predicting cryptocurrency price movements. Moreover, research in the cryptocurrency sphere largely focuses on Bitcoin, paying little attention to Ethereum, Bitcoin's second-in-line with respect to market capitalization.

In this paper, we construct hourly networks for a year of Ethereum transactions, using computed graph metrics as features in a series of machine learning models. We find that regression-based approaches to predicting Ether prices/price deltas primarily and almost exclusively rely on using current prices, motivating the need for classification models to predict price/up down movements rather than raw prices. Across a handful of such classification models, using hourly network metrics as input features, we are able to outperform baseline up/down prediction F-1 scores by up to 14%, accuracy by up to 5%, precision by up to 50%, and recall by up to 7%. These findings have implications for the future of cryptocurrency price prediction and trading activity, and suggest further research.

# ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Dan Rockmore, for his consistent patience and thoughtful suggestions throughout this research process. I'd also like to thank Professors Soroush Vosoughi and Professor Andrew Campbell for their willingness to serve on my Honors Thesis committee.

I am immensely grateful to each and every one of my Dartmouth Computer Science professors. As I look back on my CS experience, I am consistently impressed by the way this program brought me (an intimidated freshman with zero experience) in with open arms, and through the years inspired within me a sense of confidence and intellectual curiosity.

I look back on my time at Dartmouth with the utmost fondness. For the countless memories I've made, I have my incredible friends to thank.

I would like to thank my brother, whose hardworking and grounded nature has always inspired me to step outside of my comfort zone.

Finally, it is impossible for me to enumerate the ways in which I am grateful to my parents. I thank them for their endless love and support, and for each and every sacrifice they have made to get me here.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

Decentralized cryptocurrencies, created in 2009 with the goal of changing the financial-asset transfer paradigm by introducing security and anonymity, continue to capture public interest and grow in popularity. Transfer of currency ownership is done through the blockchain, a secure and verifiable system built around data validation and cryptography. Cryptocurrencies are independent of centralized authority, and the system holds the information of currency owners, units, and mechanisms to "create" new currencies. Because cryptocurrencies are fiat-based and lack intrinsic value, their prices are influenced by a variety of factors, including market risk and volatility risk (Nadler and Guo [2020]). Accordingly, transaction volume and distribution are of extreme importance when trying to understand and predict currency trading prices. For this reason, as well as the short lifespan of cryptocurrency research to date, any new insights regarding buyer/seller networks, transaction concentrations, and market participant dynamics are highly valuable and informative.

In this paper, we focus on Ethereum, a relatively new cryptocurrency whose primary coin Ether (ETH) was released to the public for the first time in 2015. Since then, Ether is only second to Bitcoin in cryptocurrency market capitalization. For Ethereum, as with most cryptocurrencies, the primary access point was and remains the ability to transfer ownership of Ether between parties on the blockchain. As a result, it is both possible and informative to analyze the transaction relationships that lie within Ethereum blockchain data. Exploring transaction relationships is a worthwhile task in terms of illustrating the way Ether is transferred between parties and, in particular, different from the patterns of equity transfers. Given the volatile and relatively unexplored nature of cryptocurrency price movements, particularly the ways in which prices may serve as a function of transaction patterns, we push our analysis of Ethereum beyond merely exploring transaction mechanisms.

In this paper, we address two main questions. First, how can we represent and visualize

the transaction network for Ethereum trading, keeping in mind the rapid and evolving nature of trading activity? Second, how can we leverage the power of the transaction network representation to inform predictions of Ether price movements and price changes?

There are several motivations for this research. First, it provides a glimpse into the way Ether is transferred across parties, enabling us to understand (in standard network analytic terms) Ether flow through hourly networks and on per-transaction levels, and the significance of top transactors when comparing incident transactions across market participants. Next, this research allows us to identify which machine learning models are useful for predicting future Ether prices, both for predicting raw prices and classifying the movement of trading prices across time periods. Third, this work allows us to seemingly establish, for the first time, a link between transaction network metrics and the prediction of future Ether trading price, identifying those features relevant for various prediction tasks.

This second motivation, predicting future prices and price movements, drives this research. Ethereum's market capitalization currently stands at 237 billion dollars, with a single unit of ETH currently trading at 1954 dollars. Over the past two years (indeed, since the timeframe of our studied data), the trading price of Ethereum has risen tenfold, indicating the increasing importance of accurate and continuously improving price indicators. For those involved in cryptocurrency (and, in general, equities) trading, the smallest improvements in the ability to predict price, as well as price movement direction, prediction, can result in significant financial gains.

# CHAPTER 2

# RELATED WORK

An abundance of research has been conducted in the area of financial product (including cryptocurrency) network analysis, as well as in product price prediction and movement classification. We examine related works systematically.

## 2.1 Financial Transaction Networks

Network-based approaches have been taken to analyze financial product prices, particularly equities, at varying levels of specificity and complexity. On the more robust end, Leibon et al. [2008] create a topological structure to explore NYSE and NASDAQ-based stock movement patterns, in doing so unraveling unique capital flow rotation patterns and identifying distinct network partitions. Their work is primarily used to understand the complexity of structure found within the equities market and the extent to which correlation structures in such systems can be understood.

At a more macroscopic, necessarily less robust, level, Dimitrios and Vasileios [2015] analyze the general stock relationship network found in data from the Greek Stock Market before and after periods of crisis. In their work, they find important distinctions between networks based on the degree of price movement correlation; moreover, they find that Greek stock networks exhibit the herd rule, whereby more connected stocks are more susceptible to large price fluctuations. Importantly, they conceptualize the characteristic of a shallow network, where actions by big actors/external factors significantly impact the composition of the constructed network.

Aslam et al. [2020] take a more comparative approach to network analysis of stocks, examining the impact of the COVID-19 pandemic on the connected stock price network established before the pandemic's beginning. They find significant contagion effects and

clustering according to geographic proximity.

It's interesting to note, when restricting our scope to equities (ignoring the case of cryptocurrencies, which we will soon explore), that transaction networks are fundamentally difficult to set up, due to the fact that such transactions occur on the open market, where user data is often anonymized and necessarily difficult to acquire. It is for this reason that any insight into transaction patterns of stock traders (abstracting the nuances between traditional equity and cryptocurrency purchasers) is relatively novel.

## 2.2 Cryptocurrency Transaction Networks

For the aforementioned reason, cryptocurrency transactions provide a rich angle with which to explore trade flow patterns. Motamed and Bahrak [2019] take a comprehensive approach to constructing transaction networks across a wide scope of cryptocurrencies, looking at Bitcoin, Ethereum, Litecoin, Dash, and Z-Cash. They find that transaction graphs are surprisingly non-assortative, meaning that transaction partners are generally property-agnostic (i.e. no selection based on partner in/out-degree). They also introduce a robust set of relevant cryptocurrency graph analysis metrics, relevantly the density, assortativity coefficient, and clustering coefficient.

Significantly, Guo et al. [2019] analyze a collection of graph metrics for Ethereum, making use of the publicly available nature of Ether transaction data. They find that transaction volume, relation, and component structure may be approximated with a power-law function, a finding which remains consistent across cryptocurrencies.

With Ethereum specifically, Wu et al. [2021] provide a useful collection of checks and metrics to construct and productively analyze transactions graphs, introducing the technique of temporal weighted digraphs, which we make use of in this research. Wu et al. [2021] also introduce the distinction between traditional blockchain and spam-address-cleaned data.

## 2.3 Machine learning and Product Prices

Perhaps the largest collection of research related to this work lies in the domain of machine learning and financial product prices. While equity-based models (as discussed in section 2.1) generally don't make use of transaction data, we nonetheless review some stock price modeling for insightful techniques.

The work of Shen et al. [2012] builds on a large volume of SVM and regression-based approaches to stock price prediction, and introduces a cross-country method using such features as national currency price and commodity prices, with the aid of SVM and decision tree techniques.

Mehtab et al. [2021] present a thorough and classic approach to the task of predicting stock returns using a wide variety of machine learning techniques. Importantly, Mehtab et al. [2021] flesh out the mechanism of using regression-based (linear, logistic, random forest, decision tree) and classification-based (logistic, k nearest-neighbor, SVM, neural network) methods, many of which we implement here.

## 2.4 Machine learning and Cryptocurrency Prices

We now explore the class of related work most similar to ours: the use of machine learning techniques for the prediction of future cryptocurrency prices/price movements. Alessandretti et al. [2018] implement a collection of basing trading algorithms (regression using prices, market capitalization, share, volume as features; regression using historical price features; long short-term memory neural networks) to compare against a baseline moving-day-average scheme. Using this, they find that supervised learning methods significantly outperform baseline price predictions; this said, it's important to note that this research was conducted at the onset of large-scale cryptocurrency trading, and therefore the extent of such baseline outperformance is likely to have significantly diminished amidst the advancement

of more nuanced trading strategies.

In support of this final qualification, Akyildirim et al. [2021] find that machine learning models using technical price/quantity indicators are able to achieve a 3% outperformance over a random baseline for ETH price prediction at the hourly level, a result which tempers our expectations of what network metrics may be able to achieve.

Greaves and Au [2015] apply the most similar overall approach to their research, exploring the predictive strength of Bitcoin transaction network metrics on future prices/price up-down movements. Notably, they introduce a relevant collection of features for such tasks, emphasizing the relative importance of the largest transactors' behavior over general network patterns. While their research is similar in scope, they focus on Bitcoin, a different currency, which has important network-specific differences (specifically, the inability to explore multiple transactions involving the same user).

This paper novelly occupies the unique cross-section between Ethereum, a relatively unexplored cryptocurrency; transaction network metrics as ML model features, a relatively unstudied approach; and it utilizes new data up to the onset of the COVID-19 pandemic.

# CHAPTER 3

# METHODS

## 3.1 Data Collection

### *3.1.1 Transactions Data*

Data documenting ether transactions was scraped from a Kaggle dataset[1], which keeps a record of all historical Ethereum blockchain transactions. The daily dataset records, for every transaction of Ether, the timestamp, value (in Wei, the smallest quantifiable Ether currency denomination), as well as the sender and recipient addresses. Due to the necessarily large nature of the 7-year dataset (563,000,000 recorded transactions), our code makes use of Google's BitQuery function to access data as needed to construct hourly graphs. For each hour of data in the selected timeframe, we query in and pull out the data for every transaction. We then store this loaded hourly transactions information in a data table. Each hour of data has on the order of 15,000 blockchain transactions.

### *3.1.2 Ethereum Price Data*

Ethereum price data was collected from a different Kaggle dataset[2], which implemented a python scraper to aggregate hourly prices from a dynamic site charting evolving cryptocurrency prices. Since our models aim to predict prices on a per-hour basis, the data points collected were limited to timestamp and opening price. These values were collected and placed alongside the outputs from our graph metric computations, to be used as the output of learning models. Due to the smaller nature of the price dataset, the full file was downloaded and loaded into our model-setup code.

1. Ether transactions data: https://www.kaggle.com/datasets/bigquery/ethereum-blockchain

2. Ethereum price data: https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory

**Data entry snapshot:**

| hash<br>Hash of the transaction | from_address<br>Address of the sender | to_address<br>Address of the receiver. null when its a contract creation transaction | value<br>Value transferred in Wei | block_timestamp<br>Timestamp of the block where this transaction was in |
|---|---|---|---|---|
| 0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060 | 0xa1e4380a3b1f749673e270229993ee55f35663b4 | 0x5df9b87991262f6ba471f09758cde1c0fc1de734 | 31337 | 08/07/2015 03:30:33 |
| 0x19f1df2c7ee6b464720ad28e903aeda1a5ad8780afc22f0b960827bd4fcf656d | 0xbd08e0cddec097db7901ea819a3d1fd9de8951a2 | 0x5c12a8e43faf884521c2454f39560e6c265a68c8 | 1.99E+19 | 08/07/2015 03:36:53 |
| 0x9e6e19637bb625a8ff3d052b7c2fe57dc78c55a15d258d77c43d5a9c160b0384 | 0x63ac545c991243fa18aec41d4f6f598e555015dc | 0xc93f2250589a6563f5359051c1ea25746549f0d8 | 5.999895E+20 | 08/07/2015 03:37:10 |
| 0xcb9378977089c773c074045b20ede2cdcc3a6ff562f4e64b51b20c5205234525 | 0x037dd056e7fdbd641db5b6bea2a8780a83fae180 | 0x7e7ec15a5944e978257ddae0008c2f2ece0a6090 | 1E+20 | 08/07/2015 03:43:03 |

### *3.1.3 Data Timeframe*

We use data from January 1, 2019, to December 31, 2019, a choice motivated by the dual goals of analyzing data as recent as possible but prior to the COVID-19 pandemic.

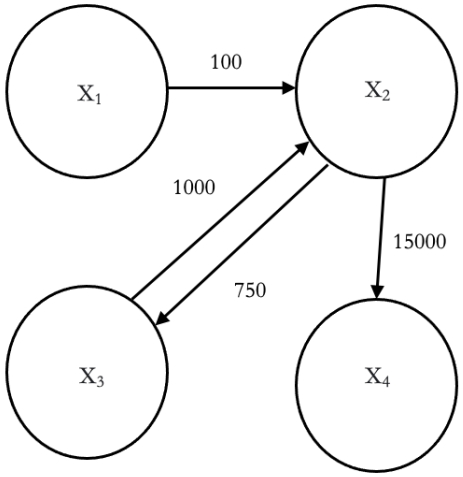
## 3.2 Building Transactions Graph

Using a combination of Python's Pandas and NetworkX packages, our code turns the data table into a weighted digraph, with nodes representing blockchain participants and weighted edges representing the value of transactions between participants. Using each hourly data table, we assemble individual hourly graphs. Notably, Ethereum's account-based construction allows us to identify repeat transaction participants through the use of account-specific (as opposed to account-transaction-specific) address hashes, in a way that other currencies do not. We make use of a slightly manipulated definition of the MTG from Motamed and Bahrak [2019] below.

**Definition 1:** ***Monthly Transaction Graph (MTG)*** *is a graph that is represented in*

*the form $MTG_n = (V_n, E_n)$ where $V_n$ is the set of nodes of this graph and each node $v \in V_n$ is an account address and $V_n$ is the set of all account addresses that appeared in the $(n+1)-th$ [hour] since beginning of the activity of that coin. $E_n$ is the set of edges of this graph where each edge $e = u, v$ shows transfer between two vertices $u, v \in V_n$.*

**Example Toy Graph:**




## 3.3 Network features/metrics

We collect a variety of features/metrics from each hourly network, with the goal of using some subset as the feature set for our models. We choose values to compute based on related literature, we define these values below.

### *3.3.1 Metrics for all nodes*

**Definition 2:** ***Total Flow*** *for an $MTG_n = (V_n, E_n)$, is the sum $\Sigma_{e \in E_n} w_e$ where $w_e$ represents the weight (value) on edge e. Total flow records the total transacted value across a graph, irrespective of direction.*

**Definition 3:** ***Number of Transactions*** *for an $MTG_n = (V_n, E_n)$ is the cardinality $|E_n|$ of the edge set in a graph. Given that each edge represents a transaction between two-parties,*

*this measure computes the total number of transactions in an hour of data.*

**Definition 4:** ***Mean Transaction Value*** *for an* $MTG_n = (V_n, E_n)$ *is the total flow,* $\Sigma_{e \in E_n} w_e$*, divided by the number of transactions,* $|E_n|$*, and represents the average transacted value (in ETH) in an hour of transactions/data.*

**Definition 5:** ***Mean Node Degree*** *for an* $MTG_n = (V_n, E_n)$ *is the total number of edges/transactions,* $|E_n|$*, divided by the number of transaction participants,* $|V_n|$*. Notably, this measure makes use of the account-based system found in Ethereum (as discussed in Section 3.2) and is increasingly informative due to the removal of empty/nontransacting nodes from our graph.*

**Definition 6:** ***Median Node Degree*** *for an* $MTG_n = (V_n, E_n)$ *is the middle value across all node degrees, and is computed by sorting the list of degrees for the list of nodes* $V_n$ *and retrieving the middle element. When transactions are heavily dominated by a few major players, this metric proves more informative than a simple average.*

In accordance with the research of Dimitrios and Vasileios [2015] and, notably, Greaves and Au [2015], we are also particularly interested in isolating the transaction patterns of the largest players in our transactions graph, specifically those transactors who are involved with a materially larger value of transacted currency than the average network participant. To this end, we first construct the following definition of a rank-$i$ graph participant.

**Definition 7:** ***Rank-i graph participant*** *for an* $MTG_n = (V_n, E_n)$ *is the node* $j$ *whose total transacted value* $t_j = \Sigma_{e \in E_{nj}} w_e$ *ranks* $i$*-th, when compared to the corresponding value* $t_k$ *for all other nodes* $k \in V_n$*. It is worth noting that the transacted-value-approach is just one metric of quantifying the "biggest participants", and existing literature also implements*

*node degree-based approaches.*

### *3.3.2 Metrics for top-ranking participants*

Given that we have formalized this definition, we now compute two metrics making use of the *Rank-i* participants for $i \in \{1, 2, 3\}$ — in plain English, the top 3 biggest participants, as is done in the related literature Greaves and Au [2015]. We define them below, keeping in mind that this choice of 3 participants is inherently arbitrary. We discuss the limitations of such a choice in our final section.

**Definition 8:** ***Total passthrough*** *for a Rank-i participant $p_i$ in a graph $MTG_n = (V_n, E_n)$ is the sum $\Sigma_{e \in E_{np_i}} w_e$ where $E_{np_i}$ is the set of all edges e such that e either flows into or out of node $p_i$. Put simply, it computes the total value of all purchases/sales for a top participant $p_i$*

**Definition 9:** ***Closeness centrality*** *for a Rank-i participant $p_i$ in a graph $MTG_n = (V_n, E_n)$ is computed as $C_{pi} = \frac{1}{\Sigma_q d(p_i, q)}$, where $q$ is a node such that there exists an edge between $q$ and $p_i$, and $d(x, y)$ is a measure of the distance between two nodes $x$ and $y$.*

### *3.3.3 Comparative cross-hour metrics*

The final class of features we extract from our graph involves the comparison of a current hour's data to the data from the previous hour. We outline them below.

**Definition 10:** ***Number of new addresses*** *between a graph $MTG_n = (V_n, E_n)$ from time $n$ and a graph $MTG_m = (V_m, E_m)$ from time $m = n - 1$ is computed as $|V_n \backslash V_m|$, representing the cardinality of the set of addresses (vertices) found at the time $n$ graph that do not transact during the hour represented in the time $m$ graph.*

**Definition 11:** ***Number of new transactions*** *between a graph* $MTG_n = (V_n, E_n)$ *from time* $n$ *and a graph* $MTG_m = (V_m, E_m)$ *from time* $m = n-1$ *is computed as* $\Sigma_{v \in V_n \setminus V_m} |E_v n|$*, representing the number of transactions (the cardinality of the edge set) involving a new vertex found at time* $n$*.*

Finally, we include the previous hour's price, price delta, and the sign of price delta as features for the relevant models, as is standard with related literature.

## 3.4 Models - Regression

We test and compare the performance of a collection of regression models for specific price forecasting on the hourly scale, making use of our aggregated data from hourly networks. We make heavy use of the Python implementations of such machine learning models found within the SKLearn packages. We outline each model below.

### *3.4.1 Baseline*

For our dummy model with which we compare the results of our more advanced predictors, we take an approach found in similar literature and apply a price increase equal to the mean of the price increases found within the training dataset. Mathematically, let's say ETH price $p_t$ for an arbitrary time $t$, on average is equal to some $p_a$ computed across the training set. To create the baseline prediction $y_i$ for the price $p_i$ for some time $i$, we merely apply the following formula: $y_i = p_a$.

### *3.4.2 Previous hour Price*

Since we expect, in accordance with Greaves and Au [2015], that the price of the previous hour holds significant predictive weight, we implement a second baseline predictor, where the price $p_i$ for an hour $i$ is simply predicted to be the price $p_{i-1}$ from the hour before. We

use this to identify and cement the fact that the previous hour's price has the most (and the vast majority of) predictive strength, a finding we use to motivate models beyond raw price prediction. Formally, we set $y_i = p_{i-1}$ for all periods $i$ in our test sample.

### *3.4.3 Linear Regression*

Linear regression fits a linear model with coefficients $\theta^T(x) = \theta_0 + \theta_1(x) + .... + \theta_n(x)$ to minimize the residual sum of squares between the observed targets in the dataset and the targets predicted by the linear approximation.

*Formulas for the n-variable linear regression:*

$$\textbf{Hypothesis: } h_\theta(x) = \theta^T(x) = \theta_0 + \theta_1(x) + .... + \theta_n(x)$$

$$\textbf{Parameters: } \theta_0, \theta_1, ...\theta_n$$

$$\textbf{Cost function: } J(\theta_0, \theta_1, ...\theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\textbf{Goal: } \text{minimize}_{\theta_0,\theta_1,...,\theta_n} J(\theta_0, \theta_1, ...\theta_n)$$

Based on standard techniques, we make 3 tuning updates to the model, implementing three forms of regularization, each described as follows.

***Ridge (L2) Regularization*** *uses the L2-norm to impose a penalty on the size of the coefficients within a Linear Regression output vector, specifically aiming to reduce the significance of overfitted data. Using SKLearn, Ridge regularization is represented by a minimization of the objective function* $||y - X_\theta||_2^2 + \alpha \cdot ||\theta||_2^2$

***Lasso (L1) Regularization*** *makes use of the L1-norm to minimize the significance of uninformative coefficients, eliminating them from the model's coefficient vector subject to the optimization objective* $\frac{1}{2n_s} \cdot ||y - X_\theta||_2^2 + \alpha \cdot ||w||_1$*, where* $n_s$ *represents the number of samples.*

***Elastic Net Regularization*** *makes use of both L1 and L2 regularization, and is therefore subject to the optimization objective* $\frac{1}{2n_s} \cdot ||y - X_\theta||_2^2 + l_1 \cdot \alpha \cdot ||w||_1 + 0.5 \cdot \alpha \cdot (1 - l_1) \cdot ||w||_2^2$, *where* $l_1$ *represents the* $L_1$ *regularization ratio found in the Elastic Net combination.*

## 3.5 Regression without Previous Price Features

We replicate our linear regressions for the same set of features, eliminating the feature that holds the value of the previous price $p_{i-1}$. We do this to understand the extent to which the previous price has a strong predictive role in predicting raw future prices, and to get a sense of the extent to which other (non-price) features can contribute to the prediction of future raw prices, exploring the coefficients associated with each other feature systematically.

## 3.6 Regression with Price Deltas

The second class of predictor models we establish aims to predict the change in price, or $\delta_{p_i}$ between a time period $i-1$ and $i$. We do this primarily to begin isolating the effects of graph features (particularly when compared to the previous hour's price, which we have established holds significant predictive weight when predicting raw prices). Further, regressing with price changes allows us to begin exploring the realm of directional price movement, a fact that is quite significant given the trading implications of buying/selling shares of ETH.

## 3.7 Evaluation Metrics - Feature Relevance

For each of our models (where applicable), we examine the relevance of each input feature, with the goal of understanding the extent to which each feature plays a role in the regression/classification task. In the results section, we display the feature coefficients for all applicable models. This metric proves useful when comparing against such baselines as previous price/previous delta, allowing us to understand the differentiated value that network

features may contribute.

## 3.8 Evaluation Metrics - Regression

### *3.8.1 MSE*

Our first evaluation metric lines up with the largest subset of existing literature, and is simply the mean-squared-error (MSE) using our prediction set. Formally,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

where $Y_i$ corresponds to the actual value for data point $i$ and $\hat{Y}_i$ is the corresponding predicted value.

### *3.8.2 Normalized MSE*

Mean squared error simply aggregates squared error across the entire sample, and is therefore not normalized to the size of the test sample values. Formally,

$$MSE_{norm} = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{|\frac{1}{n}\sum_{i=1}^{n} Y_i|}$$

where $Y_i$ corresponds to the actual value for point $i$, and the denominator represents the mean $y$ value in the sample. We take the absolute value of the mean and explore this metric when evaluating the performance of the price delta predictors, particularly to put the results in the context of the small size of our sample values.

### *3.8.3 MAD*

The next metric is simply the mean-absolute-deviation (MAD) using our prediction set. Formally,

$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n}|(Y_i - \hat{Y}_i)|$$

where $Y_i$ corresponds to the actual value for data point $i$ and $\hat{Y}_i$ is the corresponding predicted value.

### *3.8.4 Normalized MAD*

Mean absolute deviation simply aggregates error across the entire sample and is therefore not normalized to the size of the test sample values. Formally,

$$MAD_{norm} = \frac{\sum_{i=1}^{n}|(Y_i - \hat{Y}_i)|}{|\frac{1}{n}\sum_{i=1}^{n} Y_i|}$$

where $Y_i$ corresponds to the actual value for point $i$, and the denominator represents the mean $y$ value in the sample. We take the absolute value of the mean and explore this metric when evaluating the performance of the price delta predictors, particularly to put the results in the context of the small size of our sample values.

## 3.9 Models - Classification

We test and compare the performance of a collection of binary classification models (specifically predicting the up/down price movement) on the hourly scale, making use of the aggregated data from hourly networks. We heavily utilize, once again, the Python implementations of such machine learning classification models found within the SKLearn packages[3]. We outline each model below.

3. https://scikit-learn.org/

### *3.9.1 Baseline*

In accordance with similar literaure, our dummy model takes the most common classification output (a 0, corresponding to an downward price movement) as the predicted output for our data. For a time period $k$ and an overall timeframe $n$,

$$p_k = \max(\text{freq}(0), \text{freq}(1))_{t=1}^{n}$$

### *3.9.2 Probability-Based Baseline*

We implement a second baseline model, which computes the probabilities of each output class in the training set. From this, for each point in the test set, the model simply applies a probability-weighted sample and assigns the sample output to the predicted y value. We consider this analogous to a weighted coin flip, where the weights on each coin face correspond to the incidence of that face in the training set.

### *3.9.3 Previous Price Movement Baseline*

Our final baseline model simply accesses the direction of price movement from the previous time period and predicts the same value for the time period of interest. Formally, $p_k = p_{k-1}$, where $p$ holds the value of the price movement direction (1.0 for an upward movement, 0.0 for a downward movement). We do this to understand the extent to which price movements travel together, potentially indicating the existence of price movements lasting longer than an hour.

### *3.9.4 Logistic Regression-based Classification*

Logistic regression is a linear classifier, which makes use of a linear function called the logit. The logistic regression function $p(x)$ is the sigmoid function of $f(x)$, and is represented by $p(x) = \frac{1}{1+e^{-f(x)}}$. As such, it's often close to either 0 or 1, and is interpreted as the predicted

probability that the output for a given $x$ is equal to 1. Logistic regression determines the best predicted weights such that the function $p(x)$ is as close as possible to all actual responses, and maximizes the log-likelihood function (LLF) for all observations through a method called the maximum likelihood estimation. Formally,

$$\textbf{Sigmoid Function: } p(x) = \frac{1}{1+e^{-f(x)}}$$

### *3.9.5 Support Vector Classification*

A support vector machine takes classified training data points and outputs the hyperplane that best separates the points by classification. This line is the decision boundary, and for each new data point seen (i.e. test set), points are classified based on which side of the decision boundary they fall under.

### *3.9.6 Gradient Boosting Classification*

Gradient boosting classifiers are examples of ensemble learning methods, which means that they make use of a collection of other models in the learning/training process. Gradient boosting is an iterative functional gradient algorithm where each predictor tries to improve on its predecessor by reducing the errors. However, instead of fitting a predictor on the data at each iteration, it actually fits a new predictor to the residual errors made by the *previous* predictor.

### *3.9.7 K-Neighbors Classification*

K-neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Here, classification is computed from a simple majority vote of the nearest $k$ neighbors of each point: a query point is assigned the data class which has the most representatives within the $k$ neighbors of the point.

### *3.9.8 Linear Discriminant Analysis Classification*

Linear discriminant analysis[4] performs supervised dimensionality reduction by projecting the input data to a linear subspace consisting of the directions which maximize the separation between classes. Dimensions are ranked on the basis of their ability to maximize the distance between the clusters and minimize the distance between the data points within a cluster and their centroids.

### *3.9.9 Random Forest Classification*

A random forest consists of a large number of individual decision trees that operate as an ensemble, with each individual tree generating a class prediction. The class, across all ensemble trees, with the most outputs, becomes the predicted value and is contingent on the fundamental idea that *a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.*

### *3.9.10 Perceptron/Neural Network Classification*

The perceptron[5] is a linear machine learning algorithm for binary classification tasks, and unlike logistic regression, it learns using the stochastic gradient descent optimization algorithm. It takes an input data vector and multiplies data by the corresponding weights; taking this weighted sum and the bias, it inputs this value into an activation function to determine the predicted output class. The initial values for the model weights are set to small random values, and the training dataset is shuffled prior to each training epoch. This is by design to accelerate and improve the model training process.

4. https://stackabuse.com/implementing-lda-in-python-with-scikit-learn/

5. https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/

## 3.10 Evaluation Metrics - Classification

### *3.10.1 Confusion Matrix*

We can represent the output of a classification task using a *confusion matrix*, which summarizes the total number of true positives, $TP$, true negatives, $TN$, false positives $FP$, and false negatives $FN$. In the example confusion matrix 3.1, $TP = a, FP = b, FN = c, TP = d$, where the positive class corresponds to an upward price movement and the negative class corresponds to a downward price movement.

Figure 3.1: Example Confusion Matrix

| | | True movement | | |
|---|---|---|---|---|
| | | Up | Down | Total |
| Predicted movement | Up | $a$ | $b$ | $a+b$ |
| | Down | $c$ | $d$ | $c+d$ |
| | Total | $a+c$ | $b+d$ | $N$ |

### *3.10.2 Accuracy*

Accuracy $A$ is the percentage of trained samples predicted correctly, as

$$A = \frac{a+d}{a+b+c+d}$$

### *3.10.3 Precision*

Precision $P$ is the ratio of correctly predicted positive values to total predicted positive values, and identifies how often the predicted positive value is correct. In 3.1, it is represented by

$$P = \frac{a}{a+b}$$

### *3.10.4 Recall*

Recall $R$ is the ratio of correctly predicted positive values to total actual positive values, and identifies how often actual positive values are recognized. In 3.1, it is represented by

$$R = \frac{a}{a + c}$$

### *3.10.5 F-1 Score*

The F-1 score is a weighted average of precision and recall, and is represented by

$$F - 1 = \frac{2 \cdot R \cdot P}{R + P}$$

### *3.10.6 ROC Curve*

We also plot the receiver operating characteristic curve for each of our classification models. This curve shows the performance of a classification model at all thresholds. The diagonal line from bottom left to top right represents a random model (i.e. a flip of a fair coin across both output classes), and any improvement to this is represented by a curve that lies to the left/above the diagonal.

# CHAPTER 4

# DATA

Below is a table of the summary information for the network features.

Table 4.1: Data Summary (all currency amounts in Wei = $\frac{1}{10^{18}}$ ETH)

| Feature | $N$ | Mean | Median | St. Dev. | Min | Max |
|---|---|---|---|---|---|---|
| Total flow | 8471 | $8.09^{22}$ | $6.74^{22}$ | $7.29^{22}$ | $5.57^{21}$ | $2.02^{24}$ |
| Number of Transactions | 8471 | $1.69^{4}$ | $1.64^{4}$ | 5213.82 | $6.67^{3}$ | $6.2^{4}$ |
| Mean transaction value | 8471 | $4.76^{18}$ | $4.06^{18}$ | $4.24^{18}$ | $6.21^{17}$ | $1.28^{20}$ |
| Mean node in-degree | 8471 | 1.05 | 1.05 | 0.042 | 0.93 | 1.47 |
| Median node in-degree | 8471 | 0.75 | 1 | 0.43 | 0 | 1 |
| Median node out-degree | 8471 | 0.99 | 1 | 0.12 | 0 | 1 |
| Total passthrough (top 3 nodes) | 8471 | $1.74^{19}$ | $1.50^{17}$ | $2.71^{20}$ | 0 | $1.5^{22}$ |
| Closeness centrality (top 3 nodes) | 8471 | $2.80^{-3}$ | $2.19^{-4}$ | $8.7^{-3}$ | 0 | 0.15 |
| Number of new addresses | 8471 | $1.28^{4}$ | $1.24^{4}$ | 3941.1 | $4.97^{3}$ | $5.91^{4}$ |
| Number of new transactions | 8471 | $1.78^{4}$ | $1.72^{4}$ | 5644.3 | $6.94^{3}$ | $7.00^{4}$ |

*We observe that the mean transaction value is quite large, indicating that it is the effect of a few large players in the network. Similarly, the mean and median node in-degrees are very small, indicating (like with the related literature) that this graph is heavily dominated by a cluster of big transactors. Particularly when we compare the total passthrough of the top 3 nodes to the total flow, we see that (even restricting our top nodes to $n = 3$) a large portion of the transacted value is accounted for by major network participants.*

We plot three attributes for the data - number of transactions, Ethereum price, and price up/down movement. More visualizations can be found in the appendix.

## 4.1 Number of transactions

Figure 4.1: Timeframe: day

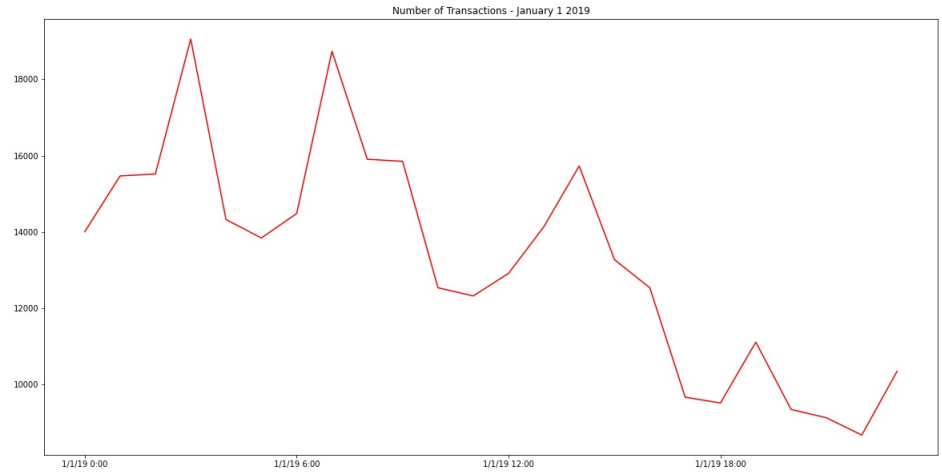
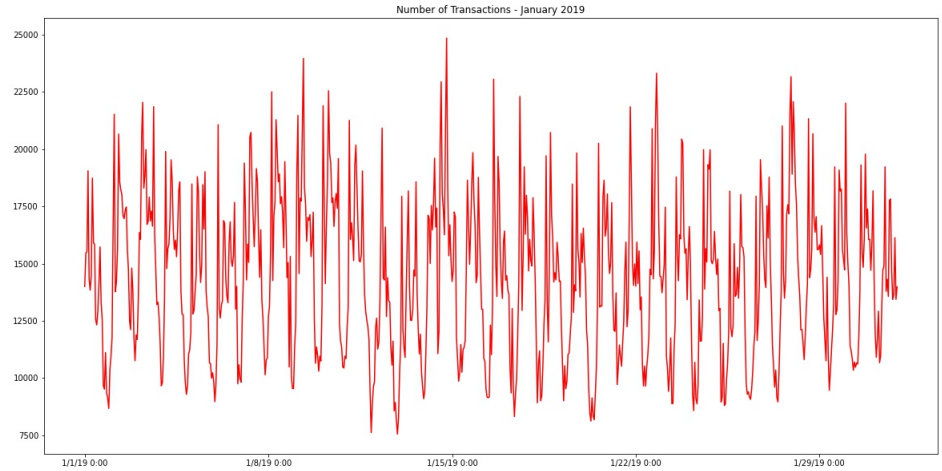



Figure 4.2: Timeframe: month



## 4.2 Ethereum Price

Figure 4.3: Timeframe: day

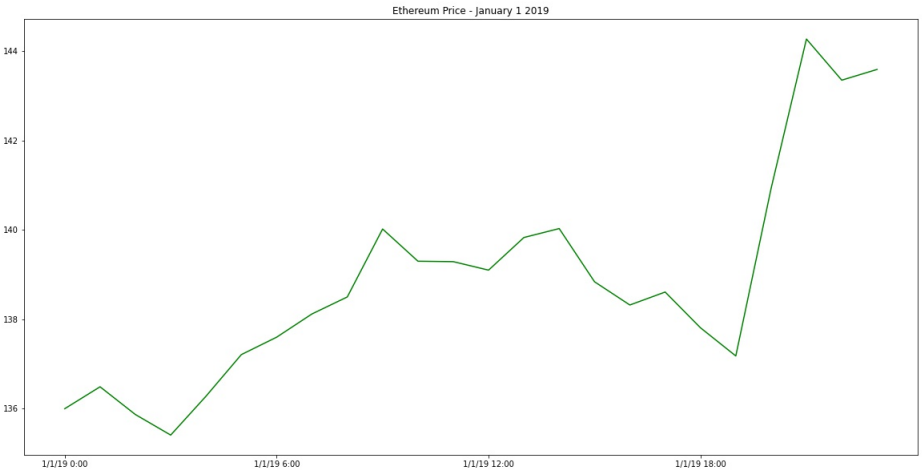
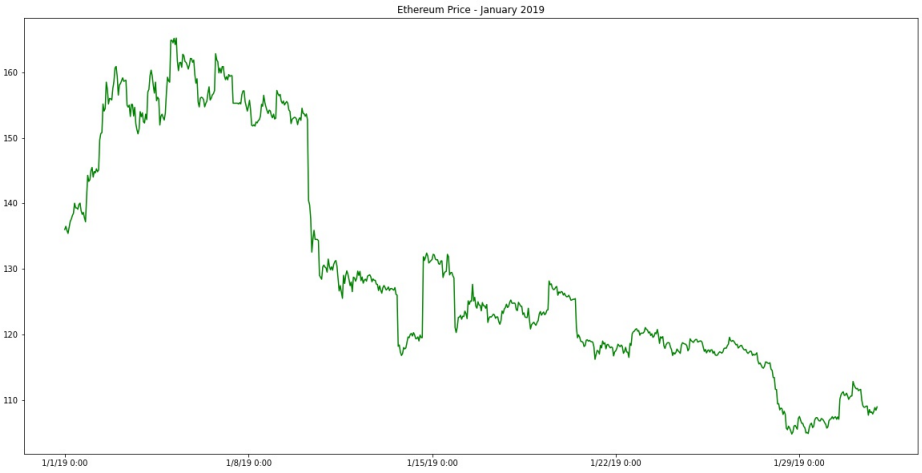
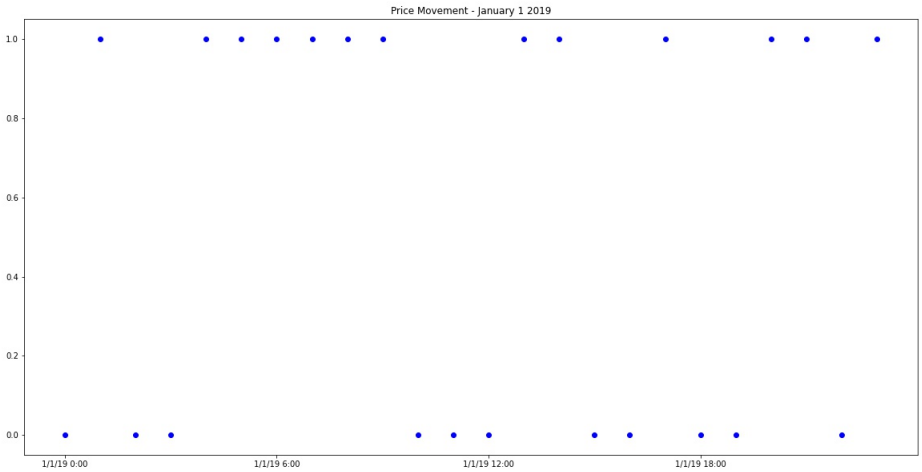



Figure 4.4: Timeframe: month



## 4.3 Ethereum Price Movement

Figure 4.5: Timeframe: day



# CHAPTER 5

# RESULTS

## 5.1 Price Regression

Our results from a collection of linear regression models are shown below.

Table 5.1: Mean-Squared Error for Price Regressions:

| Regression | MSE |
|---|---|
| Baseline | 724.1372137537435 |
| Previous Price | 1.2323796663190816 |
| Lasso | 1.2318923390725545 |
| Ridge | 1.230936344879986 |
| Elastic | 1.2318852325826923 |

*We observe that a significant portion of the MSE improvement is accounted for by simply taking a previous price, and that improvements to the linear regression model using other features are marginal.*

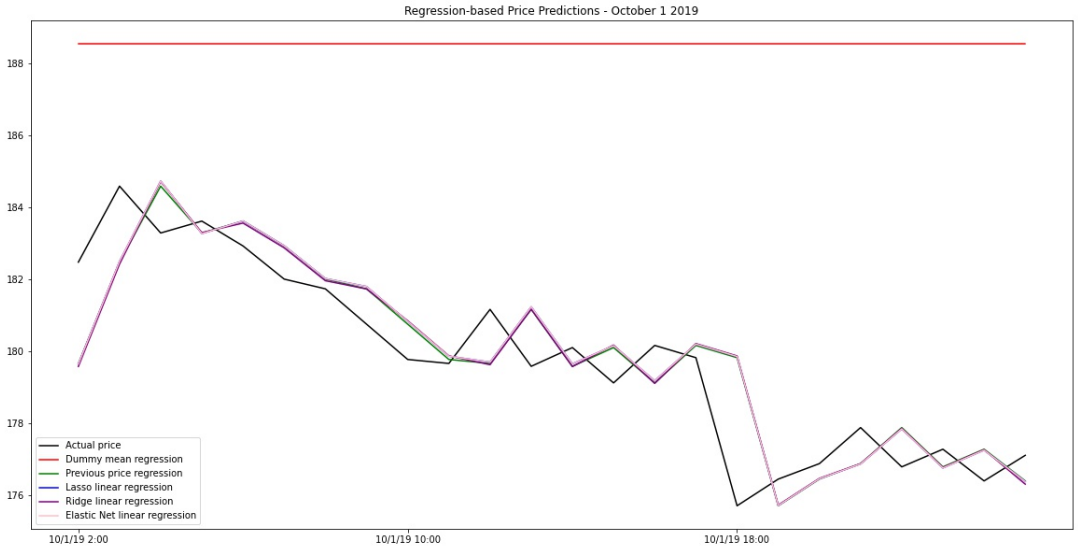
Table 5.2: Mean Absolute Deviations for Price Regressions:

| Regression | MAD |
|---|---|
| Baseline | 20.3040725999777 |
| Previous Price | 0.7224921793534929 |
| Lasso | 0.7226343987257658 |
| Ridge | 0.7212890515654306 |
| Elastic | 0.7226357611452228 |

*We observe that a significant portion of the MAD improvement is accounted for by simply taking a previous price, and that improvements to the linear regression model using other features are marginal here as well.*

Plotted predictions are below. For the sake of clarity, we only plot prediction data for the first day of the test sample (October 1, 2019). Due to the high similarity between each of the regression models, only such a time window allows for visualizable differences between the predictors.

Figure 5.1: Price Regression - Data from October 1, 2019



## 5.2 Price Delta Regression

Our results from a collection of linear regression models, regressing change in price on an assortment of features, are shown below.

Table 5.3: Mean-Squared Error for Price Delta Regressions:

| Regression | MSE |
|---|---|
| Baseline | $4.508864811344313^{-5}$ |
| Previous Delta | $9.035466545260578^{-5}$ |
| Lasso | $4.509038664778869^{-5}$ |
| Ridge | $4.5016012422187665^{-5}$ |
| Elastic | $4.50774604320323^{-5}$ |

*We observe that there are very small improvements to our baseline prediction, but that using previous deltas does not prove helpful in the way that previous prices do.*

Table 5.4: Normalized Mean-Squared Error for Price Delta Regressions:

| Regression | Norm-MSE |
|---|---|
| Baseline | 5.509575150701545 |
| Previous Delta | 11.040823807250167 |
| Lasso | 5.509787589664117 |
| Ridge | 5.5006994842901635 |
| Elastic | 5.508208079962587 |

*Looking at our N-MSE, we note that adding in network features only slightly improves our predictive value, as compared to the baseline mean value.*

Table 5.5: Mean Absolute Deviations for Price Delta Regressions:

| Regression | MAD |
|---|---|
| Baseline | 0.004330233063623503 |
| Previous Delta | 0.006615928278936393 |
| Lasso | 0.004324560862397466 |
| Ridge | 0.004316295039062451 |
| Elastic | 0.004323669472671209 |

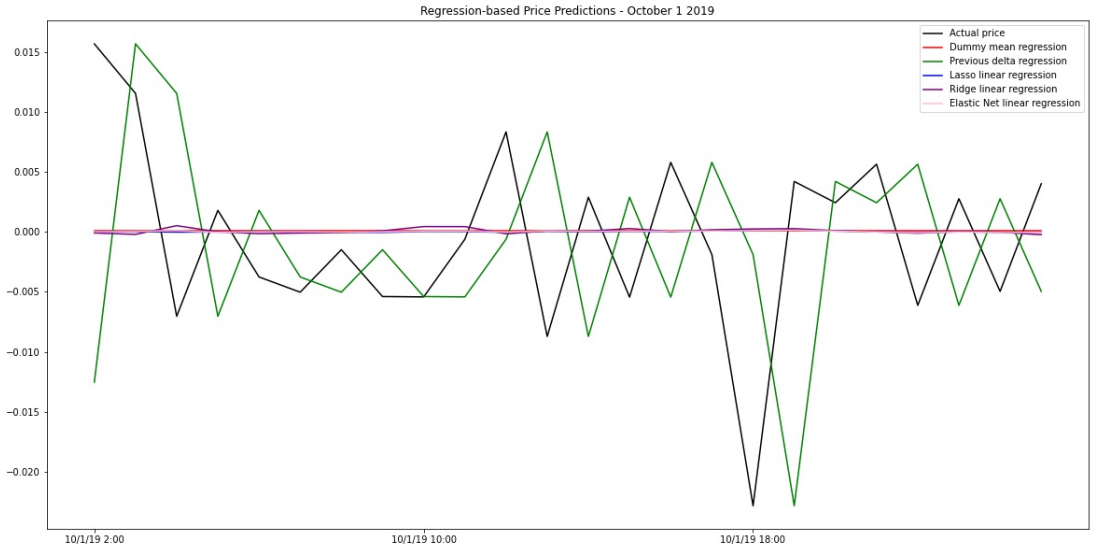
Table 5.6: Normalized Mean Absolute Deviations for Price Delta Regressions:

| Regression | Norm-MAD |
|---|---|
| Baseline | 529.1297362489145 |
| Previous Delta | 808.4286304779181 |
| Lasso | 528.4366256715886 |
| Ridge | 527.4265892932056 |
| Elastic | 528.3277029406826 |

*Looking at our N-MAD, we note that adding in network features only slightly improves our predictive value, as compared to the baseline mean value. The baseline value, however, outperforms previous delta semi-significantly.*

Plotted predictions are below. For the sake of clarity, we only plot prediction data for the first day of the test sample (October 1, 2019). Due to the high similarity between each of the regression models, only such a time window allows for visualizable differences between our predictors.

Figure 5.2: Price Delta Regression - Data from October 1, 2019



## 5.3 Price Regression without Previous Price

Our results from replicated linear regression models, removing previous price from the set of input features, are shown below.

Table 5.7: Mean-Squared Error for Price Regressions (not including previous price):

| Regression | MSE |
|---|---|
| Baseline | 724.1372137537435 |
| Lasso | 718.8783243406571 |
| Ridge | 791.005519936303 |
| Elastic | 720.5326734178254 |

*We see that after removing previous price, we are able to achieve only small (1%) improvements to reducing MSE.*

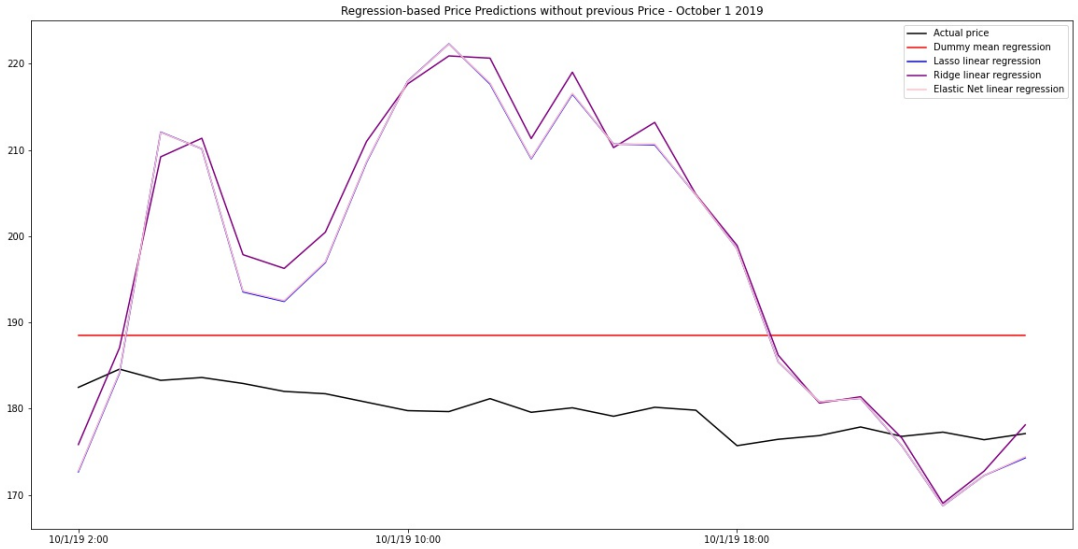
Table 5.8: Mean Absolute Deviations for Price Regressions (not including previous price):

| Regression | MAD |
|---|---|
| Baseline | 20.3040725999777 |
| Lasso | 21.5189828962225 |
| Ridge | 22.72712618707084 |
| Elastic | 21.547386323506753 |

*We see that after removing previous price, we are not able to outperform the baseline mean prediction when evaluating using MAD.*

Plotted predictions are below. For the sake of clarity, we only plot prediction data for the first day of the test sample (October 1, 2019).

Figure 5.3: Price Regression without Previous Price - Data from October 1, 2019



## 5.4 Feature Analysis for Regression Models

We display normalized coefficient tables for the best-performing regression model from each of the tasks (predicting raw prices including previous price features, predicting price deltas, predicting raw prices excluding previous price features), below.

Table 5.9: Normalized coefficients for standard price regression — Ridge

| Feature | Coefficient (Relevance) |
|---|---|
| Total Flow | 0.000 |
| Number of Transactions | 0.000 |
| Mean Transaction Value | 0.000 |
| Mean Node Indegree | 0.07293 |
| Median Node Indegree | 0.05761 |
| Median Node Outdegree | 0.16487 |
| Top Nodes Passthrough | 0.000 |
| Top Nodes Closeness Centrality | 0.02185 |
| Number of New Addresses | 0.00001 |
| Number of New Transactions | 0.000 |
| Previous Price | 0.68273 |

*We observe the relative significance of previous price on future prices, as is consistent with the literature.*

Table 5.10: Normalized coefficients for delta price regression — Ridge

| Feature | Coefficient (Relevance) |
|---|---|
| Total Flow | 0.000 |
| Number of Transactions | 0.00001 |
| Mean Transaction Value | 0.000 |
| Mean Node Indegree | -0.30573 |
| Median Node Indegree | -0.09442 |
| Median Node Outdegree | -0.42885 |
| Top Nodes Passthrough | 0.000 |
| Top Nodes Closeness Centrality | -0.31746 |
| Number of New Addresses | -0.00003 |
| Number of New Transactions | 0.000 |
| Previous Price | 0.00137 |
| Previous Price Delta | 2.14511 |

*These models as a whole did not successfuly predict future prices, so the coefficients aren't of significance.*

Table 5.11: Normalized coefficients for price regression without price feature — Lasso

| Feature | Coefficient (Relevance) |
|---|---|
| Total Flow | 0.000 |
| Number of Transactions | 0.50583 |
| Mean Transaction Value | 0.000 |
| Mean Node Indegree | 0.07293 |
| Median Node Indegree | 0.05761 |
| Median Node Outdegree | 0.16487 |
| Top Nodes Passthrough | 0.000 |
| Top Nodes Closeness Centrality | 0.02185 |
| Number of New Addresses | 0.27467 |
| Number of New Transactions | 0.21950 |

*After removing previous price from our feature, we observe that certain network characteristics begin to take on significance.*

## 5.5 Classification

Our results from a collection of classification models are shown below. We print accuracy and F-1, then plot the ROC curve for all models. For each model, we display the F-1 scores achieved by toggling the hyperparameter $h$, which represents the number of hours of prior data used as features to the model. We then plot the ROC curves obtained from the models with an $h = 1$. For all other hyperparameters, we defer to the standard SKLearn default values.

Table 5.12: Classification and F-1 Scores for various h-values:

| Classification | $h = 1$ | $h = 6$ | $h = 12$ | $h = 24$ |
|---|---|---|---|---|
| Common Value Baseline | 0.0 | 0.0 | 0.0 | 0.0 |
| Probability Baseline | 48.04 | 48.29 | 48.82 | 48.27 |
| Previous Delta Baseline | 48.3 | 48.3 | 48.3 | 48.3 |
| Logistic Regression | ***64.02*** | 39.29 | 41.36 | 45.11 |
| Support Vector | ***53.81*** | *49.20* | 43.17 | 40.36 |
| Gradient Boost | 46.49 | ***50.99*** | ***50.20*** | 45.98 |
| K-Neighbors | *49.39* | 47.45 | 49.44 | 47.26 |
| Linear Discriminant Analysis | ***51.14*** | ***52.06*** | *49.31* | 40.40 |
| Random Forest | 46.26 | 43.56 | 44.04 | 31.68 |
| Perceptron | 0.0 | 0.04 | 0.29 | ***64.99*** |

*Outperforming values are italicized, and outperforming values that beat a random baseline (50%) are bolded.*

Table 5.13: Classification and Accuracy/Precision/Recall Scores for $h = 1$:

| Classification | Accuracy | Precision | Recall |
|---|---|---|---|
| Common Value Baseline | 51.09 | 0.0 | 0.0 |
| Probability Baseline | 49.79 | 47.76 | 48.64 |
| Previous Delta Baseline | 45.09 | 43.82 | 43.86 |
| Logistic Regression | 49.90 | ***91.15*** | *49.34* |
| Support Vector | ***55.42*** | ***53.09*** | ***54.54*** |
| Gradient Boost | ***56.00*** | 39.23 | ***57.32*** |
| K-Neighbors | 50.31 | *49.60* | *49.21* |
| Linear Discriminant Analysis | ***55.16*** | *47.97* | ***54.74*** |
| Random Forest | ***55.53*** | 37.95 | ***56.78*** |
| Perceptron | 51.09 | 0.00 | 0.00 |

*Outperforming values are italicized, and outperforming values that beat a random baseline (50%) are bolded. Logistic regression performs very well, as well as SVM and LDA.*

## 5.6 Feature Analysis for Classification Models

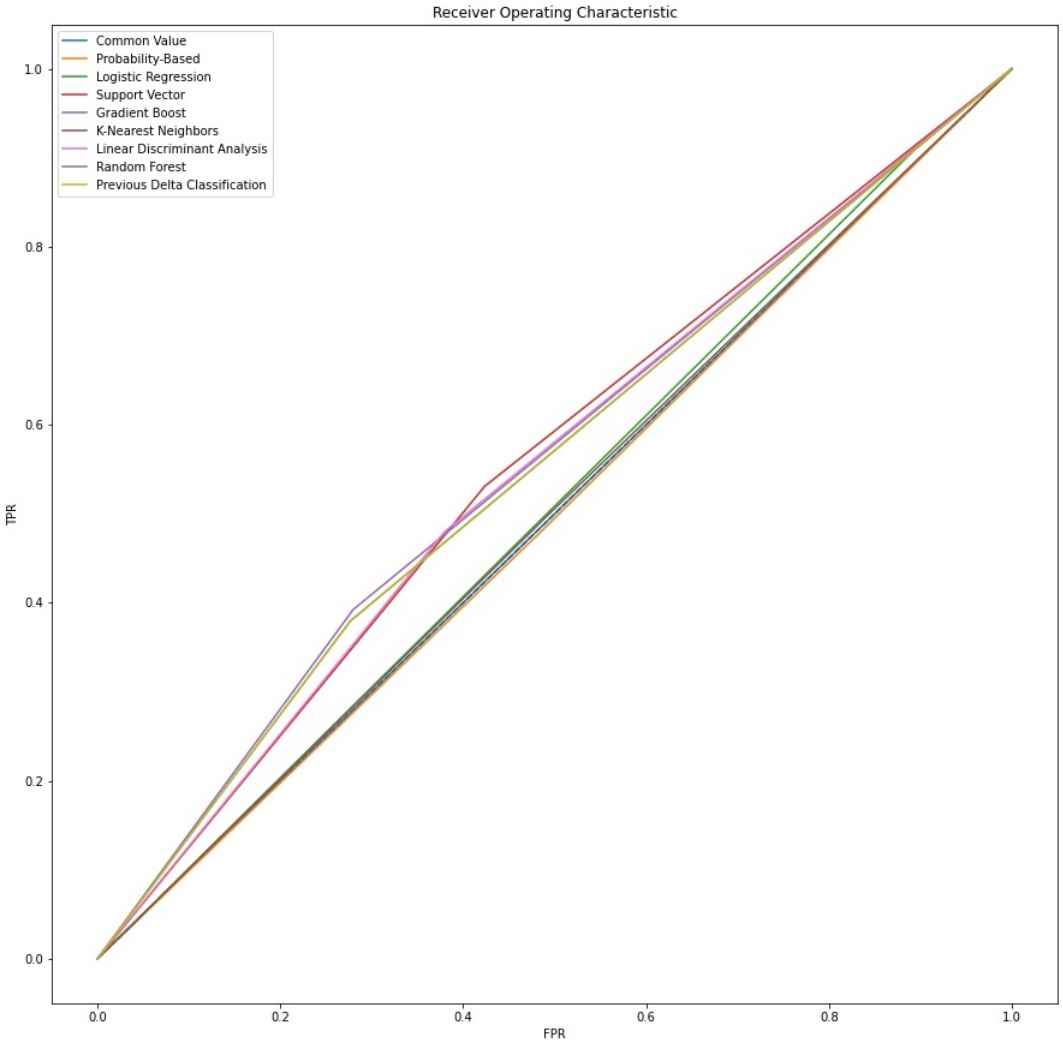
We also display the coefficients for the relevant models (i.e. ones that assign coefficients to features) below. Because we use a SVM with a non-linear kernel and K-Neighbors classification does not implement a coefficient-based system, we ignore these two models below.

Table 5.14: Classification model coefficients

| Feature | LR | GB | LDA | RF | NN |
|---|---|---|---|---|---|
| Total Flow | -0.0013 | 0.049 | 0.0 | 0.038 | 0.366 |
| Number of transactions | 0.0 | 0.085 | 0.0 | 0.028 | 0.0 |
| Mean transaction value | -0.0047 | 0.078 | 0.0 | 0.072 | -0.0070 |
| Mean node in-degree | 0.0 | 0.154 | 0.166 | 0.059 | 0.0 |
| Median node in-degree | 0.0 | 0.0074 | -0.028 | 0.0036 | 0.0 |
| Median node out-degree | 0.0 | 0.0082 | -0.069 | 0.018 | 0.0 |
| Total passthrough (top 3 nodes) | 1.00 | 0.050 | 0.0 | 0.044 | 0.64 |
| Closeness centrality (top 3 nodes) | 0.0 | 0.045 | -0.03 | 0.034 | 0.0 |
| Number of new addresses | 0.0 | 0.068 | 0.0 | 0.023 | 0.0 |
| Number of new transactions | 0.0 | 0.090 | 0.0 | 0.037 | 0.0 |
| Previous price | 0.0 | 0.11 | 0.00014 | 0.096 | 0.0 |
| Previous delta | 0.0 | 0.25 | 0.91 | 0.348 | 0.0 |
| Previous delta sign | 0.0 | $3.9^{-5}$ | 0.049 | 0.195 | 0.0 |

*We observe that, across the board, the activity patterns of the top transactors, as well as the mean node in-degree, hold predictive value.*

Figure 5.4: ROC curve for assorted classification models




*We observe that curves that lie to the left/above our baseline predictors represent improvements. As with the scores, logistic regression, SVM, and LDA all perform well.*

# CHAPTER 6

# DISCUSSION

We proceed with our discussion systematically, exploring results from each modeling task separately.

The task of regressing raw prices on an assortment of network features proved generally unsuccessful with respect to outperforming the baseline estimates, a fact we can largely attribute to the high predictive power of a previous hour's price in predicting the opening price for the next hour. Indeed, the MSE and MAD of the estimates affirmed this, as shown by the very small difference in MSE (sans normalization) found across the test set, whether or not we added network features. For a 2.5-month hourly dataset, the predicted price on average (taking the most successful model, the ridge-regularized linear regression) only missed the mark by 1.23 Wei, a successful error-minimizing prediction.

To figure out precisely how predictive previous price data is when compared against network features (for the task of predicting raw prices), we need only look at Section 5.3, where we perform the same regression having eliminated previous price data as a feature. Indeed, we see that the MSE of the models shoots up significantly, only beating the baseline (in the best case) by less than 10% (of MSE). Further, the MAD, in all cases, does worse than the baseline mean regression. Given these results, we can reasonably conclude that there is little predictive information available from the constructed networks when exploring raw prices. When looking at Figure 5.9, which decomposes the strength of each feature by displaying the associated coefficients, we see that previous price has a significantly higher normalized coefficient, accounting for the majority of the predictive information. We would be remiss to ignore the presence (albeit weak) of the coefficient for mean node out-degree, which may work in tandem with previous price and account for the marginal improvement to price baseline predictions. When exploring coefficients for the regression without previous price (Table 5.11), we see that network features begin to take on significance, with coefficient increases

seen across the feature set (notably number of transactions and new participants). The positive relationship between higher network metrics and upward price movement checks out with our expectations of increased trading activity relating to price ascent. We speculate that, without previous prices as a feature, network features take on more significance primarily using the transaction value and the metrics that compare graphs across hours. If this is the case, there may be some predictive value in quantifying the impact of new entrants to the transactions space.

While a fair bit of trading activity is focused around the estimation of prices, changes in price often hold much more significance. Indeed, even if the error is a fraction of a cent off, predicting price in the wrong direction could have significant implications in terms of buying/selling product. For this reason, we place heavier emphasis on understanding hour-to-hour price movement, a task whose two approaches, regression and classification, we analyze now.

Predicting price changes proved to be a much more interesting task, particularly with respect to the importance of previous hour data. Firstly, because our price dataset was at the hourly level, there was only so much granularity available with respect to compare price information across hours (particularly when Ether trades at the sub-second level). Instead of using previous prices as a baseline, we instead looked at the predictive strength of simply taking the previous hour's delta, a method that reflects the fact that directional price movements often occur together and last often many hours ("spikes" or "free-falls"). Interestingly enough, this pattern was not reflected in the data, and simply taking the mean delta across the training dataset proved more predictive than replicating the previous hour's delta (perhaps largely due to the lower variance and corresponding error seen here). This regression showed a slight sign of the predictive power of network metrics, as indicated by the lower MSE/norm-MSE/MAD/norm-MAD for the ridge and elastic net regression models, which implemented a collection of features. When looking at Table 5.10 — the normalized coefficients for the price delta regression — we notice that more features begin to take on

significance, but that, interestingly, an upward price delta prediction is negatively related to increased mean and median node degrees, as well as top node closeness centrality. The best guess we have for an explanation is that increased participation (centrality) from the top players (a mechanism which is also reflected in the transaction volume of smaller players and, accordingly, their node degrees) may be indicative of rapid selloff movement, which is interpreted by the market as indicative of imminent price fall. This said, these results are generally weak and heavily dominated by the previous price delta.

When exploring the visualized data, we notice that all of the models cling tightly to a zero value for delta, indicating that movement-magnitude wise it proves much stronger to predict no movement than to predict movement and risk capturing the wrong direction. It is for this reason that the ability to classify movement more definitively (rather than just deferring to an expectation of zero delta) is all the more interesting.

This classification task, as anticipated, proved much more informative. With respect to the hyperparameter test, barring a few exceptions (gradient boost and LDA, notably) we observe that an increase in the number of hours of past data explored adds little (potentially negative) predictive value, and therefore we largely focus our analysis on the case of $h = 1$. We further make this judgment call because the magnitude of baseline outperformance seen in non $h = 1$ cases is limited. Particularly when we perform a cost-benefit analysis of time and computation space versus performance success, we largely favor the $h = 1$ case. It is worth pointing out that with an increase in the time window of the data, the perceptron significantly outperforms all other classification models.

Exploring our results, we find significant improvements in the classification task using the logistic regression, support vector machine, and linear discriminant analysis approaches, each of which makes use of previous hour prices and an assortment of network features. Most notably, the logistic regression model is able to classify with an F-1 of 64.02, a significant improvement to the probability baseline of 48.04 (and to the best baseline, a random up/down prediction, with a score of 50.00). With respect to accuracy, support vector, gradient boost,

and LDA classification were also able to achieve material outperformance of the common value baseline, in the best case (gradient boost) beating the baseline by 5%. In terms of precision, the logistic regression model performed very well, achieving an impressive precision of 91.15 (alongside high Accuracy and Recall scores, it is worth noting) — this finding is perhaps the most significant from the lens of predicted upward movements being correct. Indeed, for a market participant looking for a good time to buy Ether, the high precision rating can mean a confident approach to predicting when trading prices will go up. With respect to recall, a handful of models achieved very good results, in the best case (the gradient boost algorithm) achieving a recall score of 57.32. This finding is similarly notable, particularly from the angle of correctly recognizing when prices will move up.

When exploring the coefficients, we observe that different classification models take significantly different approaches to placing weights on our features. Across the board, however, total passthrough of the top 3 nodes seems to be indicative of price upward movement, in accordance with similar findings from Guo et al. [2019].

This research is very promising in terms of the significance of transaction pattern metrics on prediction and understanding of price movement, particularly in the context of cryptocurrency, which are known for rapid transaction movement and high-speed ownership transfers. All things considered, however, the significance of previous price demonstrated through regressions, as well as the generally bounded nature of the ability to outperform baseline predictions, seems to indicate that the movement of Ethereum prices may also be significantly tied to macroeconomic/time-specific factors, a fact which may nonetheless be connected to transaction patterns.

## 6.1 Limitations + Further Research

There are a few key limitations to this approach, in terms of the structure of data, as well as the feature set we use to model. Due to the mismatch in available timeframes between the transactions dataset (per minute or per hour) and the price dataset (per hour), we only analyze price movement on a per-hour basis. Cryptocurrencies, like most traded financial products, are subject to very rapid price movements and fluctuations, and as a result, simply looking at hourly fluctuations obscures a lot of available information in more real-time contexts. In these cases, we expect previous time-window prices to be equally, if not more, predictive, but nonetheless immediate transaction behaviors may be very indicative of price movements in the immediate aftermath, especially when considering such market events as sell-offs or short squeezes. Indeed, when looking at figure 4.3, we see that each hour is characterized by semi-significant price movements. To the extent that collected data can help narrow this price movement down, the robustness of predictions can be improved. With respect to feature selection, this research is limited in the sense that we ignore the impact of features beyond the narrow scope of transactions data, failing to acknowledge the influence of such macroeconomic factors as world events, cryptocurrency and cryptocurrency exchange news, and the ability to mine new cryptocurrency units. Particularly when examining the interaction between such features and the evolution of our transactions network, there is likely room for material improvements to classification model performance.

A final limitation has to do with the choice, as discussed previously, of selecting the top 3 performing nodes for our top-participants metrics. Indeed, this is in accordance with the work of Greaves and Au [2015], but given the way transaction value is distributed across graph participants (the fact that a large majority is being transacted by the top participants, with very little involvement from smaller players), a more informative approach may be to take the subset of participants that constitute some $k$ percentage of transactions and utilize their information. Further, this $k$ could be a hyperparameter, tuned on even a smaller timeframe, to maximize the predictive value extracted from the network asymmetry.

These limitations frame the discussion in terms of further research, particularly work that focuses on improving the granularity of data and set of interacting input features. This study ends prior to the start of the COVID-19 pandemic, attempting to understand the patterns of transaction data/cryptocurrency prices during normal times. When tracking the movement of financial markets, and, notably, cryptocurrency prices today, we see that times of crisis introduce a whole new set of circumstances to grapple with when predicting price movements. Informative further research could isolate predictive models to various pre/during/post-crisis time frames, decomposing the relevance of input features across such time horizons. Finally, in the same way this research builds on the work of Greaves and Au [2015] with respect to analyzing Ethereum, there is lots of work to be done across the cryptocurrency sphere, and similar analyses for sibling cryptocurrencies will likely prove equally, if not more, insightful.

## 6.2 Conclusion

As the cryptocurrency landscape evolves, each axis with which we can conduct analysis adds a dimension of clarity to what remains a complex and unexplored landscape.

While this study was necessarily limited in the scope of data analyzed and used for modeling, we were able to find interesting patterns in terms of transaction network evolution and significant predictability of price up/down movement from network metrics. Already, these models outperform baseline predictions in material ways, and it is our hope that the task of sourcing new price-prediction-features (particularly as such network metrics, where helpful, diminish in their usefulness) remains of paramount importance to those interested in making or understanding successful trading decisions.

# REFERENCES


Erdinc Akyildirim, Ahmet Goncu, and Ahmet Sensoy. Prediction of cryptocurrency returns using machine learning. *Annals of Operation Research*, 297:3–36, 2021.

Laura Alessandretti, Abeer ElBahrawy, Luca Maria Aiello, and Andrea Baronchelli. Anticipating cryptocurrency prices using machine learning. *Complexity*, 2018:1–16, nov 2018.

Faheem Aslam, Yasir Tariq Mohmand, Paulo Ferreira, Bilal Ahmed Memon, Maaz Khan, and Mrestyal Khan. Network analysis of global stock markets at the beginning of the coronavirus disease (covid-19) outbreak. *Borsa Istanbul Review*, 20:49–61, 2020.

Kydros Dimitrios and Oumbailis Vasileios. A network analysis of the greek stock market. *Procedia Economics and Finance*, 33:340–349, 2015.

Alex Greaves and Benjamin Au. Using the bitcoin transaction graph to predict the price of bitcoin. 2015.

Dongchao Guo, Jiaqing Dong, and Kai Wangc. Graph structure and statistical properties of ethereum transaction relationships. *Information Sciences*, 492:58–71, 2019.

Gregory Leibon, Scott Pauls, Daniel Rockmore, and Robert Savell. Topological structures in the equities market network. *Proceedings of the National Academy of Sciences*, 105(52): 20589–20594, 2008.

Sidra Mehtab, Jaydip Sen, and Abhishek Dutta. Stock price prediction using machine learning and LSTM-based deep learning models. pages 88–106, 2021.

Amir Pasha Motamed and Behnam Bahrak. Quantitative analysis of cryptocurrencies transaction graph. 2019.

Philip Nadler and Yike Guo. The fair value of a token: How do markets price cryptocurrencies? *Research in International Business and Finance*, 52, 2020.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Shunrong Shen, Haomiao Jiang, and Tongda Zhang. Stock market forecasting using machine learning algorithms. 2012.

Jiajing Wu, Jieli Liu, Yijing Zhao, and Zibin Zheng. Analysis of cryptocurrency transactions from a network perspective: An overview. *Journal of Network and Computer Applications*, 190, 2021.

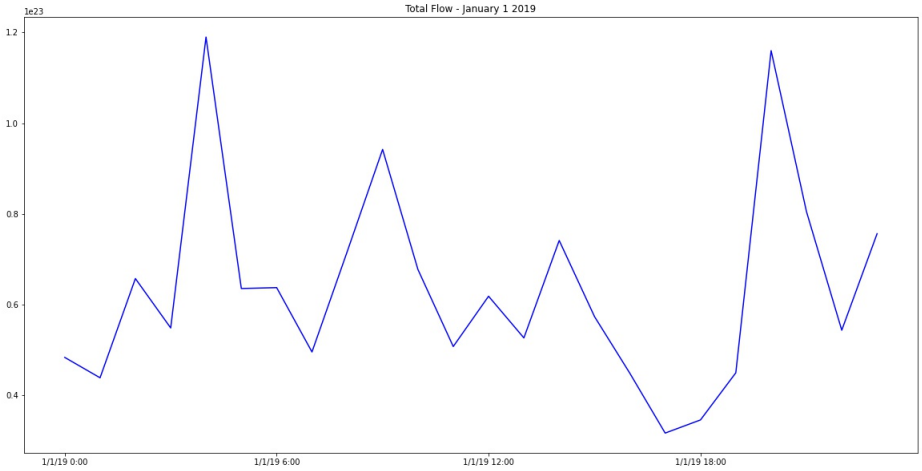
Tony Yiu. Understanding random forest: How the algorithm works and why it is so effective. 2019.

# APPENDIX A

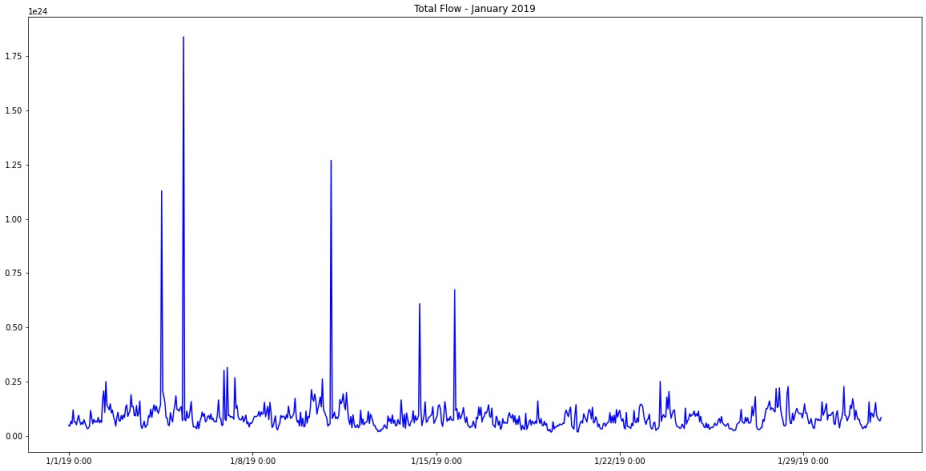
# FEATURE PLOTS

## A.1 Total Flow

### *A.1.1 Timeframe: day*

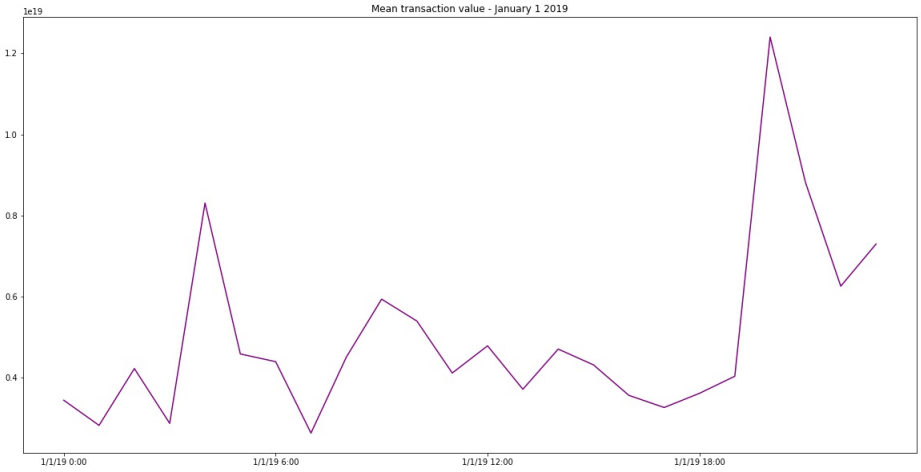



### *A.1.2 Timeframe: month*

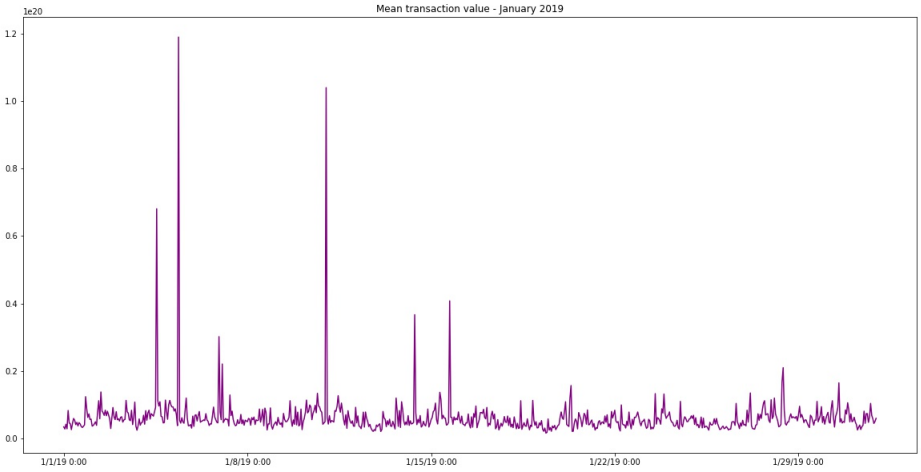


## A.2 Mean transaction value

### *A.2.1 Timeframe: day*

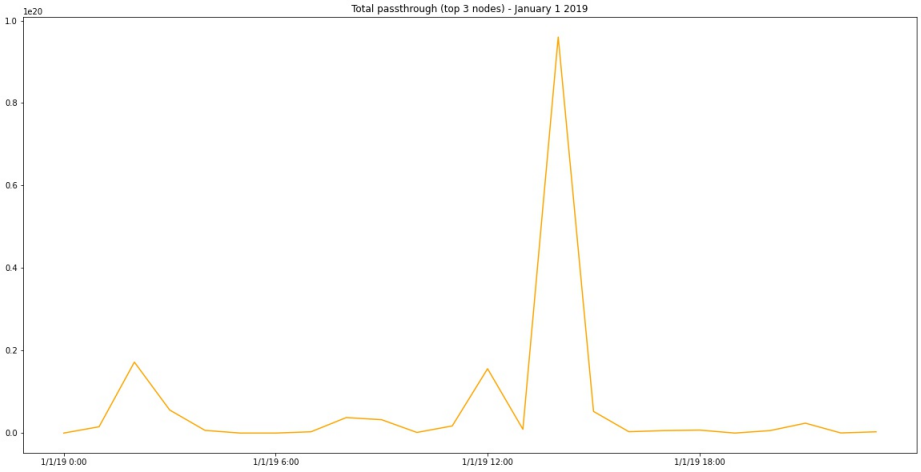



### *A.2.2 Timeframe: month*

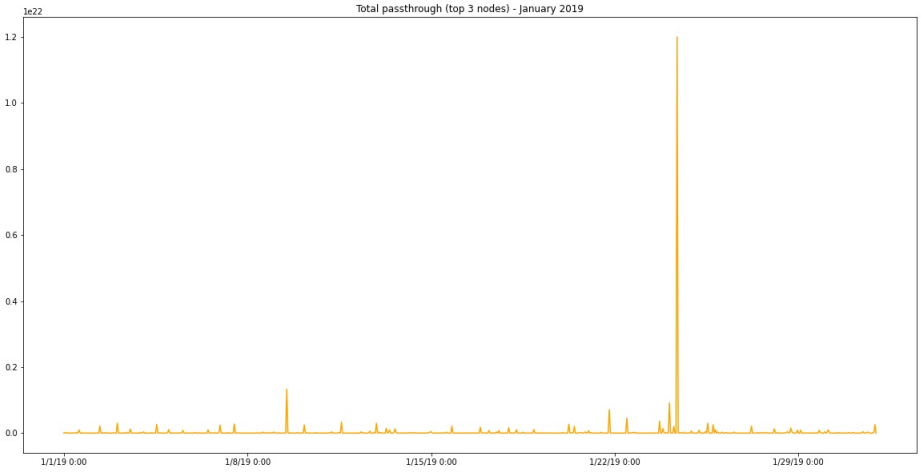


## A.3 Total passthrough - top 3 nodes

### *A.3.1 Timeframe: day*

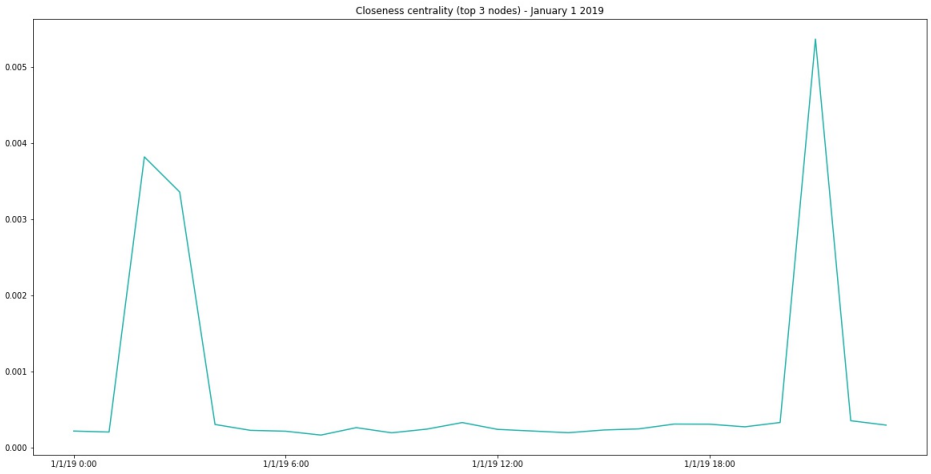



### *A.3.2 Timeframe: month*

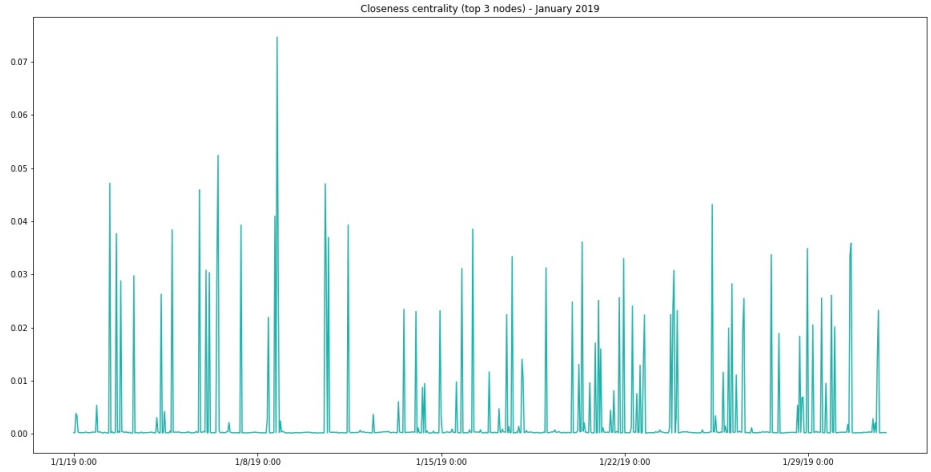


## A.4 Closeness centrality - top 3 nodes

### *A.4.1 Timeframe: day*

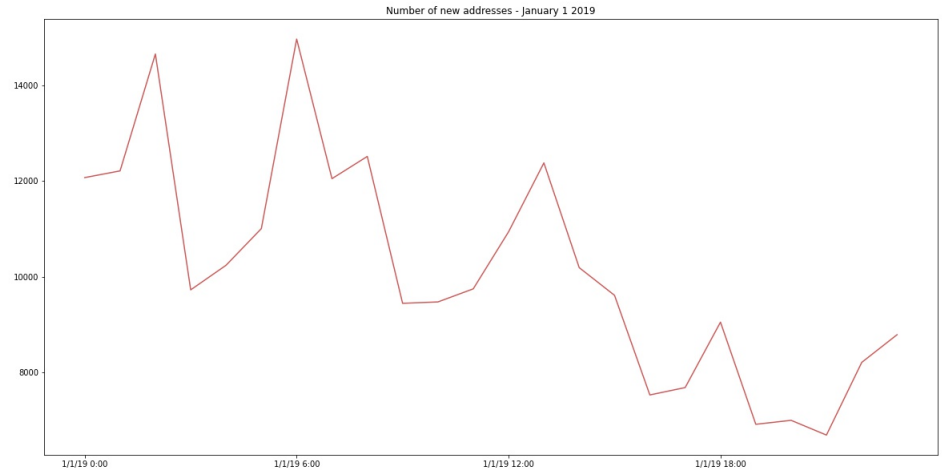



### *A.4.2 Timeframe: month*

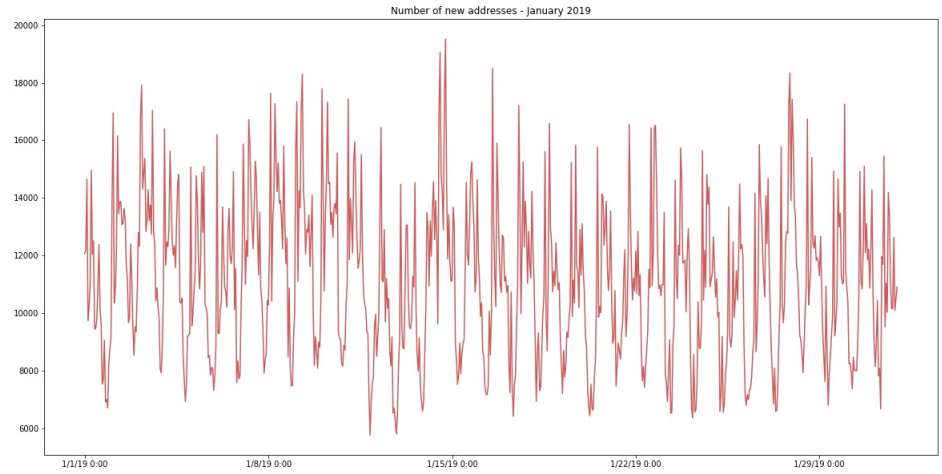


## A.5 Number of new addresses

### *A.5.1 Timeframe: day*

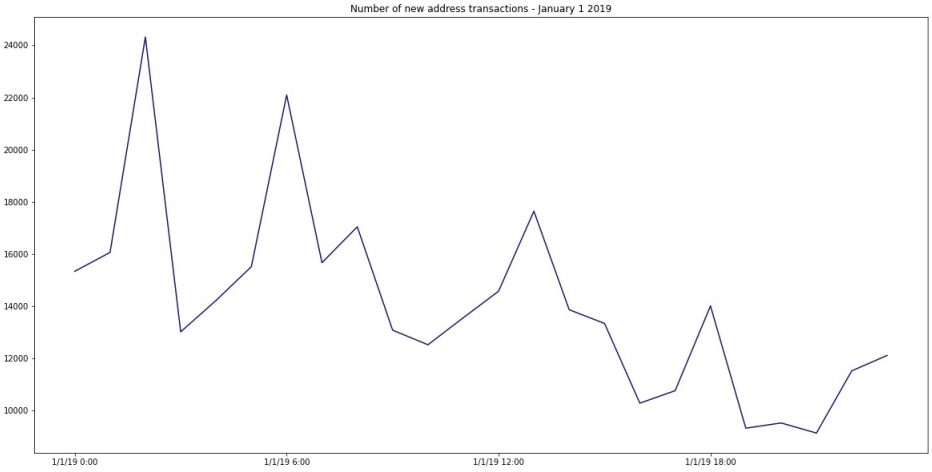



### *A.5.2 Timeframe: month*

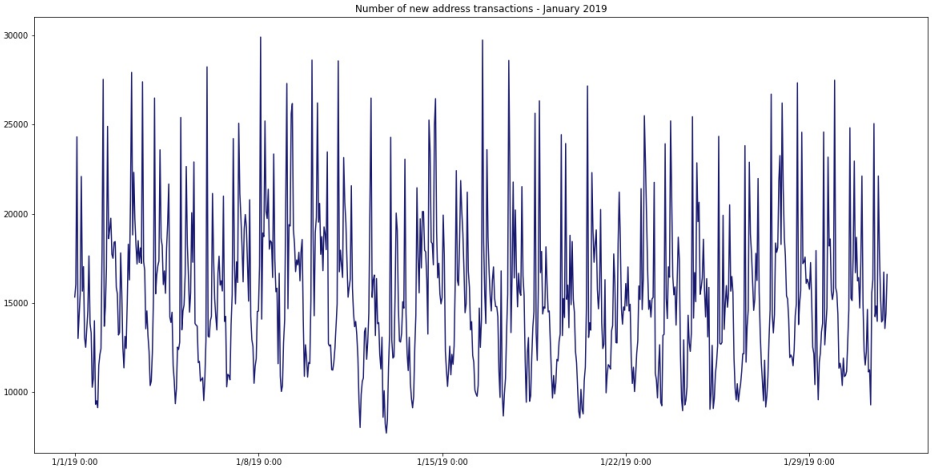


## A.6 Number of new address transactions

### *A.6.1 Timeframe: day*

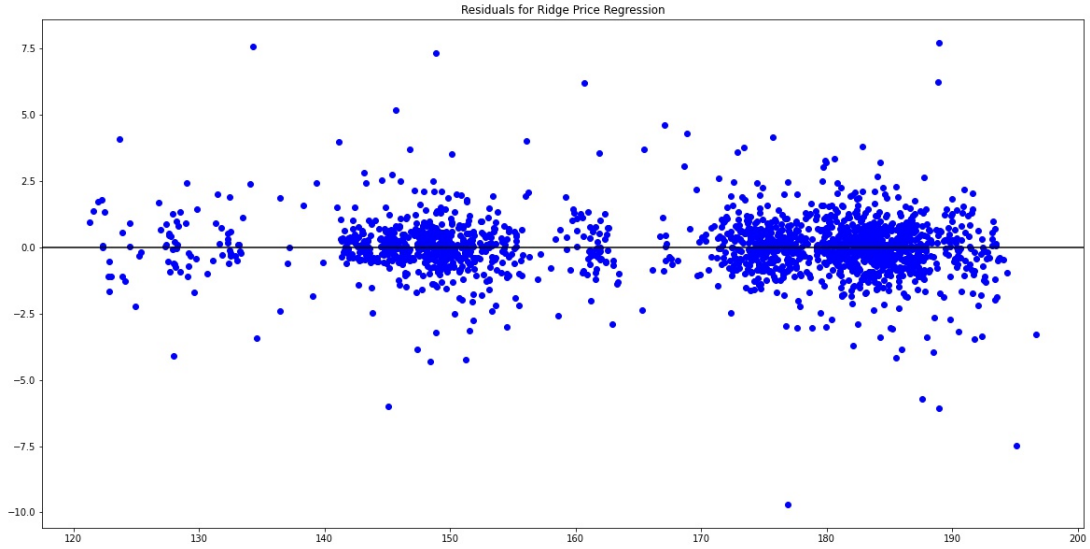



### *A.6.2 Timeframe: month*



# APPENDIX B

# RESIDUAL FOR LINEAR REGRESSION PREDICTION — RIDGE




*We observe a residual plot where the y-values are clustered around zero, evenly distributed.*

# APPENDIX C

# PRICE REGRESSION WITH LOGGED FEATURE VALUES

Table C.1: Mean-Squared Error for Price Regression with logged features:

| Regression | MSE |
|---|---|
| Baseline | 724.1372137537435 |
| Previous Price | 1.2323796663190816 |
| Lasso | 45.5261465250828 |
| Ridge | 38.91704880229184 |
| Elastic | 532.6538658068824 |

*We observe that taking the log of our feature values actually diminishes the performance with respect to MSE.*

Table C.2: Mean Absolute Deviations for Price Regressions with logged features:

| Regression | MAD |
|---|---|
| Baseline | 20.3040725999777 |
| Previous Price | 0.7224921793534929 |
| Lasso | 6.467496081118304 |
| Ridge | 5.611526362134112 |
| Elastic | 18.331372290426255 |

*With respect to MAD, we see that logging our features once again diminishes model performance.*

# APPENDIX D

# TRADING SIMULATION USING SVM CLASSIFICATIONS

The final task implemented used the results from our SVM classifier, the best overall performer, and took a simple approach to a trading simulation. The rules were as follows (and assume that at any point, buying/selling is possible — necessarily abstracted).

1. Retrieve SVM-predicted price up/down movement
2. If the price is predicted to move up, buy 1 unit and sell in the next time period
3. If the price is predicted to move down, sell 1 unit and buy back in the next time period

---

```
balance = 0
for i in range(len(svc)):   iterate through classified predictions
   movement = svc[i]
   if movement == 1:   up movement
      balance += (prices[i+1] - prices[i])
   elif movement == 0:   down movement
      balance += (prices[i] - prices[i+1])
```

---

Across our 3-month test period, following this simple algorithm yielded a final balance of **$176.09**, implying the marginal, yet notable, financial gains of such research. Had our algorithm correctly predicted movement every time, the profit would have been **$1437.35**. A lot more work is needed to fully capture this difference, but nonetheless our model holds some notable predictive significance with real-world relevance.