Cryptocurrency Pumping Predictions: A Novel Approach to Identifying Pump And Dump Schemes

Cameron Ramos, *Student, Department of Computer Science, Stanford University*
Noah Golub, *Student, Department of Computer Science, Stanford University*

Abstract—We present a model that can identify artificial increases in crypto-currency prices (called 'pumps') from publicly available order data. Our dataset consists of 179 24-hour order books from several prominent currencies. These order books are split into two sub-sets: nominal market movement order books and abnormal rate change order books. We first trained a SVM model for baseline results. We then developed Neural Net models. They are able to recognize pumps with 81.245% accuracy. More interestingly, they can not only recognize, but also predict if a pump will occur in the next 12 hours with high accuracy: 82.5%.

1 Introduction

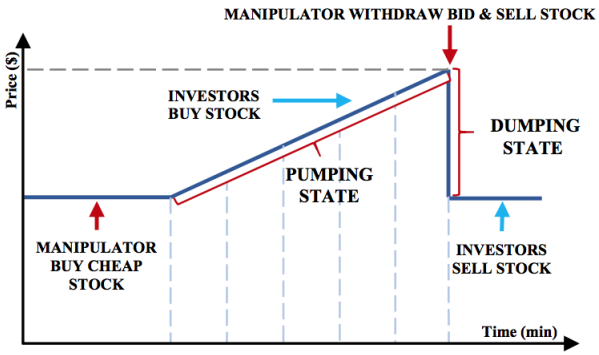
Pump and dump schemes are the bane of any ambitious crypotrader's existence. Pump and dump refers to the process of a bad actor artificially raising the price (the pump) and subsequent selling to make a large profit (dump). While traditional stock markets have legal safeguards that prohibit pump and dump schemes, the wild west of crypto has no such assurances, and are far more common in this domain. Successfully detecting pump and dump schemes will help prevent trading losses during the "dump" phase of the cycle. Traders also stand to make a lot of money from the pump phase of the cycle, in which market book activity manipulates traders into believing that the price of an asset (or cryptocurrency in this case) is higher than it should be.

Importantly, during the pumping phase, the manipulator artificially increases demand by issuing large buy orders at the market price of the currency being traded. These massive buy orders force other participants on the exchange to purchase the currency at a slightly higher price or risk not having their orders completed. This edges the market price higher and higher. The manipulator rarely actually makes a purchase during the pump phase since they continually cancel their buy orders just before they are filled. Eventually, the bad actor decides to issues massive sell orders very rapidly (thus executing the dump).

We propose price-agnostic models for recognizing and predicting artificial price increases, meaning that the models are trained on order book data without the bid or ask rate for a particular cryptocurrency. This is desirable because all cryptocurrencies have different maximum supplies and token economics, so it is difficult to normalize all coins to a standard price. The input to our models are 24-hour blocks of orders containing the type of order (buy or sell), the volume for a set of N orders. Each order is also tagged with an index representing its time relative to other orders (1 to N). Every training example is tagged with whether or not it represents an anomalous price increase (above 15%). We then use an SVM and a Neural Network to output a prediction on whether the market data represents an anomalous price increase. For our predictive model, we only feed our Neural Net 12 hours of market data and then evaluate if it can correctly predict if the full 24 hour order book represents an anomalous price increase or normal market movement.

2 Related Work

In our proposal we identified a paper from the 2016 International Conference on Advanced Computational Intelligence **which presents** novel ways to detect pump-and-dump schemes in traditional equity markets, and did so with 88% accuracy. The paper describes how bad actors typically execute the pump and dump, summarized in the figure they provide below:



The researchers of this paper investigated the characteristics of stock price manipulation. They studied two manipulation models: pump-and-dump and spoof trading. Spoof trading is a procedure to trick other investors that a stock should be bought or sold at the manipulated price.

For their models, they used "level 2" market data (consisting of market metadata like trading volume, highest bids, lowest bids, etc.), and used them to generate a labeled training set consisting of buy/sell orders similar to the training sets our group assembled directly from the exchanges we used. Importantly, level 2 market data includes order cancellations, which are important indicators for price manipulation. From this cancellation data, it was possible for the research group to label their dataset. They

then implemented feedforward neural network models from "level 1" data, containing more trade specific information (individual trade volume, rate, etc., but no information about order cancellation). Level 1 data is more accessible to investors as it is readily available from most exchange APIs. The group's neural network model achieved 88.28% for detecting pump-and-dump schemes but failed to model spoof trading.

This research presented the broad framework for our own application as its principles are generalizable to any liquid market. We too used level 2 data to label our dataset but trained and evaluated our models with level 1 data.

3 Dataset and Features

After creating API keys for Poloniex, a widely used crypto exchange, we built infrastructure to request, filter, and compress publicly available order data between USD and a variety of other crypto-currencies: 'ETH','XMR', 'DASH', 'XRP', and 'LTC'.

Poloniex provides a JSON of up to 50,000 orders for a given crypto-pair between two unix timestamps. Initially, we manually identified obvious pump and dumps and then called our scraping function to fetch relevant order data. Because this was infeasible to collect a large training set, we decided to scrape any 24 hour region in which a crypto asset went up 15% or more in price compared to Bitcoin. We also collected order data from regions of stable market prices as obvious non pump-and-dump regions.

Once we had collected over 200 training examples, we then filtered the data. Each order in our JSON response contained three fields: the time it was placed, the type of order (buy/sell), and the rate w.r.t. USD that the asset was bought/sold at. Because our model must be time invariant, but the order of each transaction matters, we decided to replace the time of transaction field with the index of the transaction in the JSON.

Because each 24 hour window represented by our datapoints had different numbers of orders, we used Bresenham's line algorithm to identify N evenly spaced intervals in which we could average orders in between these intervals. The algorithm works by determining the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. Selecting evenly spaced points output by the algorithm gives us the indexes we average our order data between. This averaged representation thus contained: an index from 0,...,$N-1$, the proportion of buy orders in the interval, the average volume of each transaction. We originally included the average rate each transaction was executed at, but decided to later remove this from our dataset. Any data point with less than N orders was discarded.

To preserve our data without having to recompute the filtered dataset, we compressed each datapoint and built infrastructure to uncompress the full dataset when the model is trained. Using our data-scraping procedure, we procured over 250 data points and split them into two evenly sized training and test sets. We trained our models for different values of N to test the effect of higher order book granularity on model accuracy.

4 Methods

The difficulty of solving our problem is that we have no access to cancel order data from the API offered by the exchange we monitor, Poloniex. Instead, we need to find a way to detect artificial pumps using an asset's order book, which is a public ally available set of buy and sell orders with transaction volumes. Left with just this order book as our feature space, we have massive amounts of unstructured data that must somehow figure out how to surmise whether anomalous price spikes are occurring without direct access to the smoking gun cancel order data.

Initially, we implemented a Support Vector Machine model. SVM's are supervised learning models that establish a multidimensional hyperplane that separates labeled datapoint which are treated as n dimensional vectors. We implemented this method using the python sklearn module. This module provides functionality for training and evaluating a C-Support Vector Classification model. This is a slight extension of an SVM in that it includes a penalty parameter, C, that can be used to increase the weight of a misclassification. Thus, it is possible to adjust the multidimensional hyperplane so that it has more optimal classification over its training set rather than optimal separation between data points of different labels. A mathematical basis for this algorithm is provided below. Given training vectors X in two classes, and a vector Y, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$
$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, ..., n$$

After getting baseline results, we took a clue from the paper we reference, and decided to implement a standard feed forward neural network. This model works by establishing a series of connected "neurons" with associated activation functions that are applied to a linear combination of the neuron's inputs and weight vector (including intercept term). The model is "trained" by performing backpropegation, an iterative update method that updates each neuron's weight vector via gradient descent. The gradient is the partial derivative of some loss or cost function C with respect to any weight w (or bias b) $\delta C/\delta w$. As more an more training examples are processed, the model's weights approach an optimal value. Our neural network contains one input layer of size N, an initial hidden layer of the same size, and two additional hidden layers of size 20 and 2 with sigmoids as activation functions for all. This model is unique in that it's structure is a function of the resolution of the input data. In cases where data has been averaged into coarse buckets ($N = \{300, 400, 500\}$), the model is smaller and trains more quickly.

To get predictive results, we made a second flavor of the Neural Net model with an initial hidden layer of $N/2$ nodes which was trained on only the first half of each datapoint (representing the first 12 hours of market data). The accuracy of the model, however, was assessed against level 2 data of the whole 24 hour period (ie. could the model correctly predict the price movement of the second half of the dataset).

Fig. 1. SVM 24 hour model

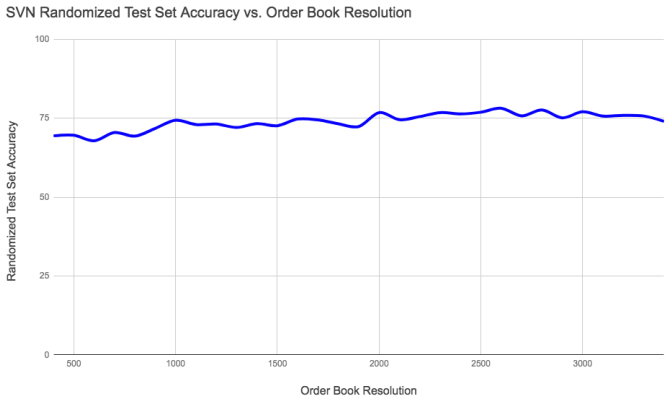
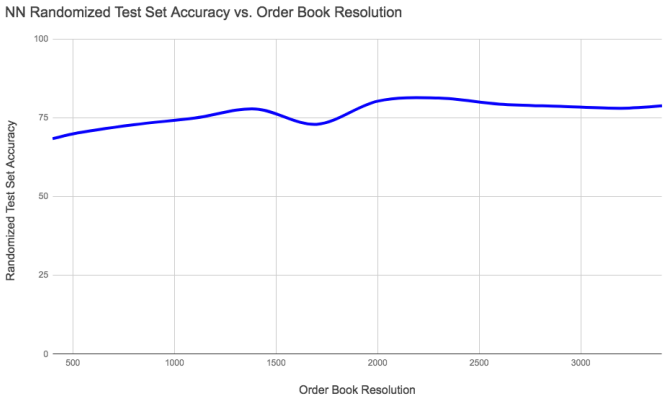


Fig. 2. NN 24 hour model



For the rest of the paper we refer to this model as the 12 hour Neural Net model.

Because neural network models have proven to be successful in equity markets, we believed this would translate well to crypto markets. Furthermore, we identified the primary challenge of our initial implementation to be the collection, filtration, and compression of vast amounts of market data. Being able to use off the shelf learning tools like Python's tensorflow, were thus attractive as the actual back-propagation updates to train our model would not have to be implemented by hand and we could focus our efforts on gathering good data.

5 Results

We evaluated our model in several ways. First, we set $N = 500$ and trained our models on a randomized half of our total dataset. Accuracy of the model was determined by comparing model classification to true labels for the half of the dataset that was not included in the training set. This was repeated in a loop 16 times to get averaged results:

Model	Training Accuracy	Testing Accuracy
SVM (24 hour)	95.18	69.578
NN (24 hour)	72.37	69.88
NN (12 hour)	70.88	69.36

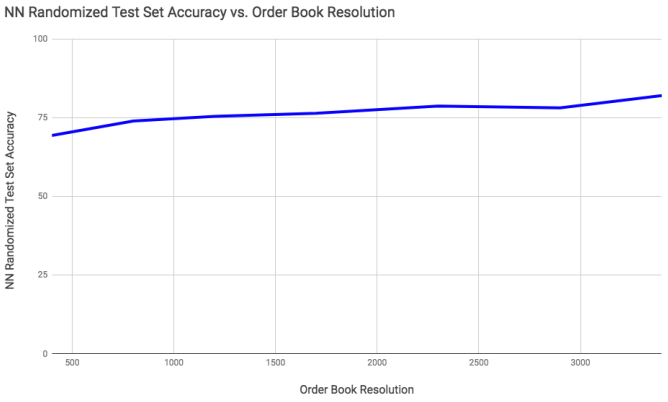
We then retested at a higher order book resolution $N = 1000$:

Model	Training Accuracy	Testing Accuracy
SVM (24 hour)	95.666	74.34
NN (24 hour)	80.6	74.897
NN (12 hour)	75.41	73.77

Because the performance of our models was dependent on the resolution of the input data, it was necessary to run this analysis for various resolutions and pick the resolutions for each model that performed best. Furthermore, we noticed that higher data resolution reduced the total size of our dataset since we were throwing away order books with less than N total orders. Fig 1, 2, and 3 show the accuracy of each model as a function of the order book resolution.

Selecting the best resolutions for each model we achieved our final results:

Fig. 3. NN 12 hour model



Model	Train Acc. %	Test Acc %	N
SVM (24 hour)	94.6	78.13	2600
NN (24 hour)	82.8	81.245	2300
NN (12 hour)	78.2	82.5	3400

6 Discussion

Ultimately, we found that it is possible to predict 24 hour price movement with a limited view of the order-book, showing that simple features such as volume and the type order are strong indicators of market trends. Results were better than expected, especially for the SVM which cannot capture complex feature relationships. A higher resolution order book (with some exceptions) tended to improve results. This was true even though higher resolution order books reduced the size of our dataset. Despite the fact that higher resolution order books were more accurate, even low resolution order books yielded positive results. Lastly, we found that we were also able to predict whether the price of a crypto-asset would increase over 24 hours after only training on the first 12 hours. This sort of model thus lends itself to algorithmic trading strategies that trade based on the expectation of the price increasing.

7 Future Work

If given 6 more months to work on this project, there are a few more improvements we would make. First, we would

attempt to track order cancellations. Our dataset does not track whether orders are canceled, a feature that strongly indicates if a pump and dump is occurring. We could either attempt to model cancellations as a latent variable with some probably distribution over the likelihood of cancellation with respect to the average order of the past day, or simply find a better API with order cancellation data. We could also attempt to incorporate our model into a trading strategy. Our model can be used in an MDP like trading strategy that chooses actions based on confidence that asset price.

8 CONTRIBUTIONS

1) Cameron built out the Poloniex data gathering and compression infrastructure for level 1 data, as well as the data normalization code that converts the JSON to a labeled 2-d numpy array that can be fed into the model generation code. There are some convenience features in the code to prevent re-pulling and re-normalizing the same data. He also implemented the SVM model and the control loop that trains and evaluates both models over multiple runs.
2) Noah compiled the list of pump and dump schemes that should be scraped (at first manually, and then by using the threshold increase code that automatically identified obvious pump and dump regions). He build out the code to automatically assign labels to data regions based on level 2 market data. He also implemented the NN model generation code using Python's Tensorflow package.

9 REFERENCES

1) T. Leangarun, P. Tangamchit and S. Thajchayapong, "Stock price manipulation detection using a computational neural network model," 2016 Eighth International Conference on Advanced Computational Intelligence (ICACI), Chiang Mai, 2016, pp. 337-341.