Predicting Cryptocurrency Fluctuations Based on Public Opinion

Dante Zakhidov[1] (zakhidov), Abdulmalik Obaid[1] (amobaid), Scott Keene[1] (stkeene)
[1]Department of Materials Science and Engineering, Stanford University
Github: https://github.com/obaiam11/Bitcoin-Price-Prediction

Introduction:

Cryptocurrency has recently taken off with Bitcoin increasing in value by ~13x in the past year. However, one of the risks associated with this currency is the fluctuations in price, which can be as large as a 24% change in price in a single day[1].Most quantitative methods used to analyze markets use numerical data but struggle to accurately account for the effect of speculation and emotions. In this work, we aim to build a neural network which can predict market trends by using public confidence in Bitcoin in order to strategically analyze Bitcoin trends.

Dataset:

We first started with Bitcoin market data that was publicly available on Kaggle[2]. The dataset consists of Bitcoin historical data from December 1st, 2014 to January 8th, 2018 divided into one-minute increments. This time frame consists of 1,574,274 minutes. For each timestamp, the data included information on the opening value, the closing value, the highest value, the lowest value, the volume traded, and the weighted price. In an effort to iterate quickly and build an initial model, we opted to first analyze the polarity trends in the market. The dataset was labeled as true if the price went up at the end of the minute timestamp and false if it stayed the same or decreased.

$$Up\ Label = \begin{cases} True, & Close - Open > 0 \\ False, & Close - Open \leq 0 \end{cases}$$

By training on the trend of the market, we hoped to be able to predict when the price would change. In addition to adding the "up label" which is our y column, we added columns for the variance and frequency of our data. After seeing that a basic polarity metric doesn't work, we bin the data into buckets of positive or negative percent increase to reduce dependence on noise.

Looking at a trimmed version of the dataset, we see from the first few rows that there will be issues with distribution of the dataset. The price of bitcoin was stable in 2014 and 2015 as interest for the cryptocurrency had not yet gained traction. The price then almost monotonically increased until late 2017 where it plummeted and that ends the data set. This volatility in the cryptocurrency market (and any market) dominates our dataset and is the main hurdle in accurate prediction.

	Timestamp	Open	Close	Up_Label
0	1417411980	300.0	300.0	False
1	1417412040	300.0	300.0	False
2	1417412100	300.0	300.0	False
3	1417412160	300.0	300.0	False
4	1417412220	300.0	300.0	False

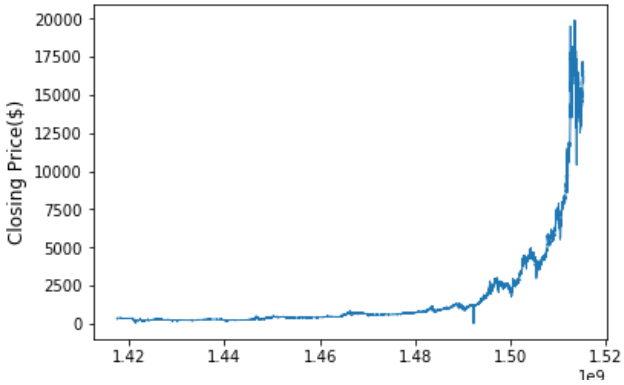


Figure 1: Trimmed dataset showing Unix timestamp, open and closing values, and how we labeled the change. The figure on the right shows Bitcoin price from Dec 2014 to Jan 2018 with the x-axis in unix time.

First Pass: Predicting Polarity using RNN

The obvious end-goal of creating a cryptocurrency based neural network is to predict price fluctuations in real time. With this goal in mind, we were eager to start with a highly temporally resolved dataset. If we could get information on a minute by minute or second by second timescale, we could do an even better job of predicting prices and staying ahead of the market. Furthermore, there would be millions of data points and that would be a dataset size that neural networks excel at. However, as we alluded to above, we realized that there also issues with highly resolved data. When looking at our minute dataset, we had an intuition that there would be no change on a minute timescale, or if there was change, that it was very small and noisy. The graph above shows that nearly all the 1.5 million minutes fall within the "third bin" which represented price changes below 0.003%. As a result, our model wouldn't be able to learn the price change since it would mostly be fitting to noisy data and any meaningful change would be drowned out. To note, at this early point in the project we had not actually binned our "y-values" yet but based on our intuition, we decided to convert the minute dataset into a daily dataset. The graph above is made after the fact to show the distribution.

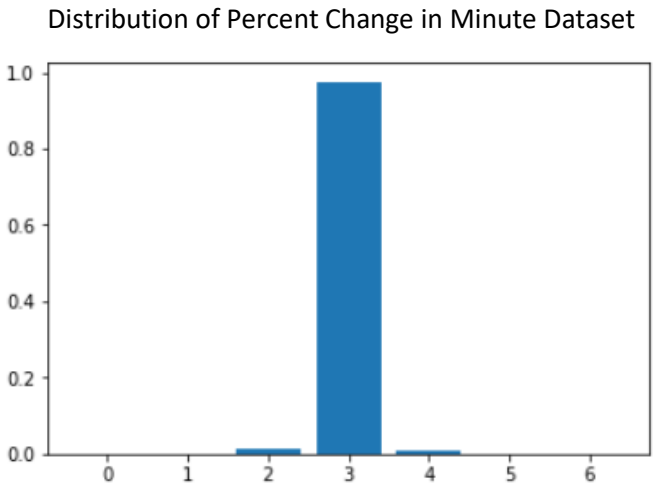


Figure 2: X-axis represents bins showing percent change. The middle bin (3) has the majority of the 1.5 million minute data points and represents change that is -0.003 %< change < 0.003%

Given the temporal nature of the data, it was most intuitive to use a recurrent neural network. We decided to start with a long short-term memory (LTSM) model. We had three LSTM layers followed by a dense layer with batch normalization in between. We used a 4 day moving window and had a softmax classifier to categorize the output.

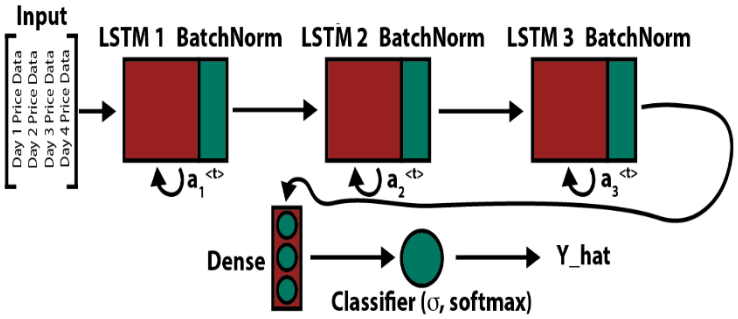


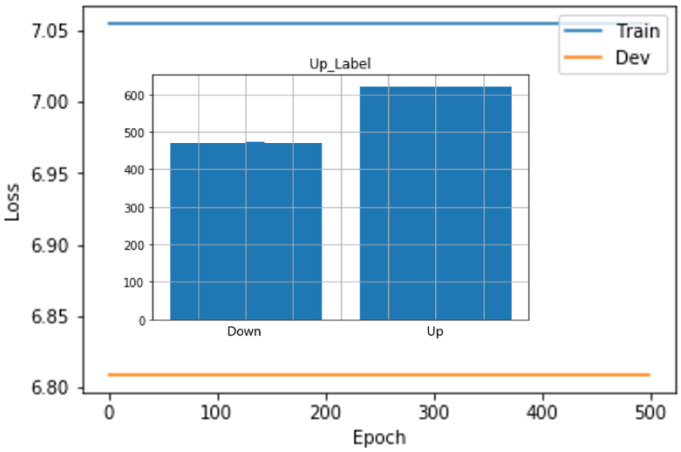
Figure 3: RNN model and parameters

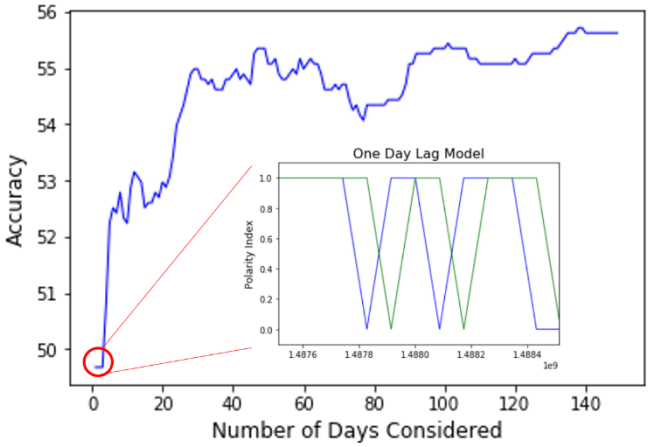
RNN Model Settings

Layer (type)	Output Shape	Param #
batch_normalization_53 (Batc	(32, 4, 6)	24
lstm_40 (LSTM)	(32, 4, 64)	18176
batch_normalization_54 (Batc	(32, 4, 64)	256
lstm_41 (LSTM)	(32, 4, 64)	33024
batch_normalization_55 (Batc	(32, 4, 64)	256
lstm_42 (LSTM)	(32, 64)	33024
batch_normalization_56 (Batc	(32, 64)	256
dense_14 (Dense)	(32, 7)	455

Total params: 85,471
Trainable params: 85,075
Non-trainable params: 396

However, even after converting to the less noisy daily dataset, looking at just polarity changes, we quickly saw that our model was unable to fit even the training set. In the figure below on the left we see that the loss for the train and dev sets doesn't change at all over 500 epochs and the accuracy is 55.8%. This means that the model finds a stable minimum immediately. If you look at the distribution of the "up" and "down" labels (remember, not binned at the moment), we see that the number of up labels divided by total days is equal to 55.9%. So, the model just guesses "up" each time instead of developing a pattern. Furthermore, if you compare to a baseline model that uses exponentially weighted moving average, we see that the baseline accuracy hovers around 55% depending on the number of previous days considered. It was clear at this point that learning on a simple parameter such as up or down was not sufficient.

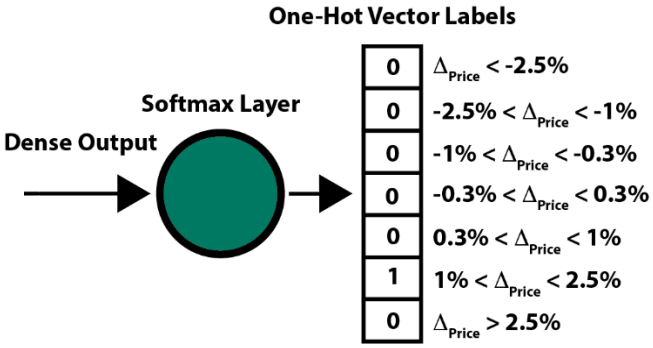


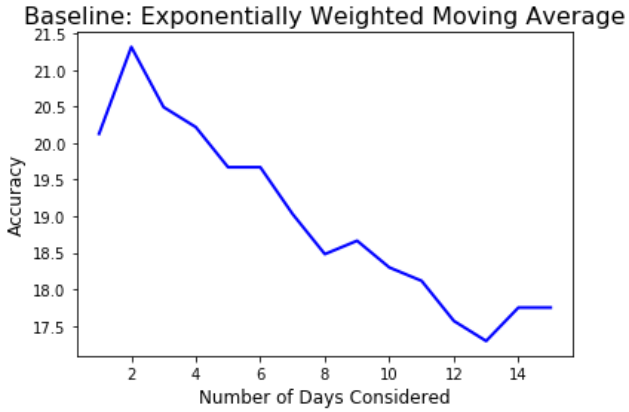


Figures 3 and 4: Figure on the left shows inability of our model to train on a polarity score. Figure on the right is a baseline model to assess minimum accuracy that needs to be beat. It uses an exponentially weighted moving average. When the number of days considered is one, we have a simple lag model where yesterday's polarity is used to predict the polarity for today.

Analyzing Percent Change: Overfit Training but no Generalization

Based on a good suggestion from our TA Guillaume, we moved away from learning on polarity to learning on percent change. This was calculated by taking (Closing cost – Opening cost)/Opening Cost. A one-hot vector was created for each time point and correctly classified based on percent change. We spent considerable time figuring out how to bin the percent change to make an even

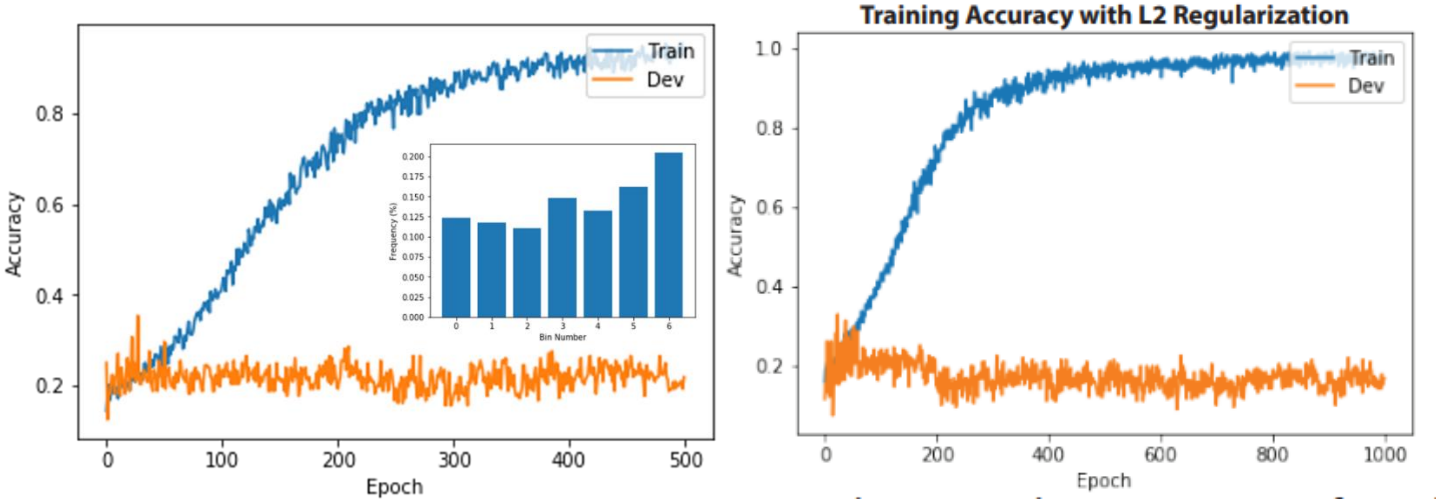




Figures 5 and 6: Figure 5 shows the new classification scheme and Figure 6 shows the corresponding baseline

distribution. We first tried to make logarithmic bin spacing but the distribution was uneven. Then we created a function where we could iterate through many different bin settings and ultimately decided on 7 custom bin ranges that are shown in Figure 5. This created a relatively even distribution of values that can be seen in the inset of Figure 7. Given the seven bins, random guessing would give a 14.3% accuracy. Looking at the exponentially weighted moving average baseline, we see that we need to beat a ~21% accuracy

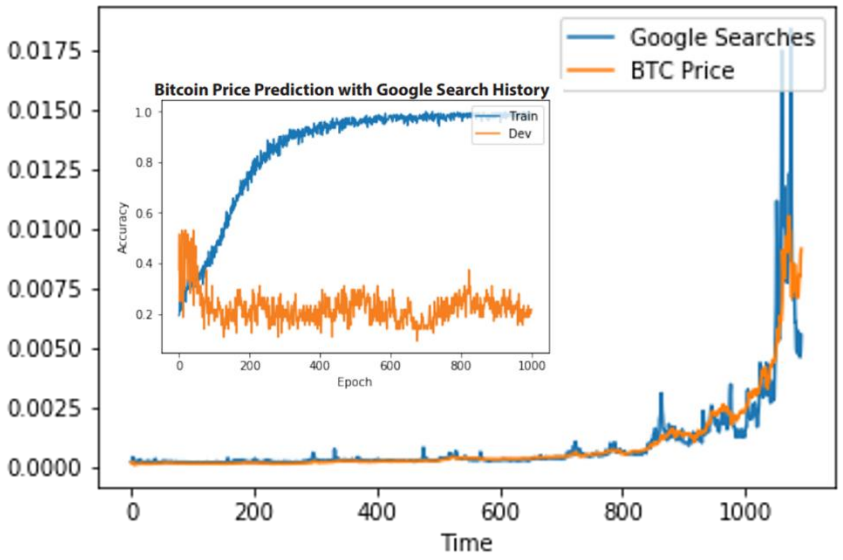
Training our LSTM model on this new set of classifications, we see in Figure 7 that we can overfit on the training set but are still not able to generalize well to the validation set. The validation accuracy can not beat our baseline. Seeing that there is a large variance problem, we look to regularization to try to improve our accuracy. We tried dropout, L2 regularization, and L1 regularization but none of them improved the validation set accuracy. L2 worked the best, allowing us to fully overfit the data but the validation accuracy stayed at ~20%.



Figures 7 and 8: Figure 7 shows model performance on binned y-values Figure 8 shows effect of L2 regularization

More Data: Adding Google Search Data and Twitter Sentiment

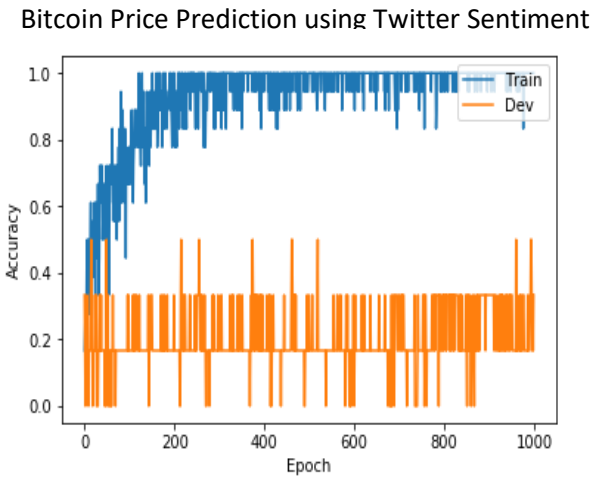
At this point, we see that pricing data alone can not accurately predict changes in bitcoin price and external data is needed. We first look to google search history and see if we can use number of google searches to predict prices. Inputting a normalized daily search dataset from Google[4] as another variable, we don't see any improvement our dataset. We still overfit our training data and our validation accuracy hovers just above 20%. We look at the overlap of google searches with

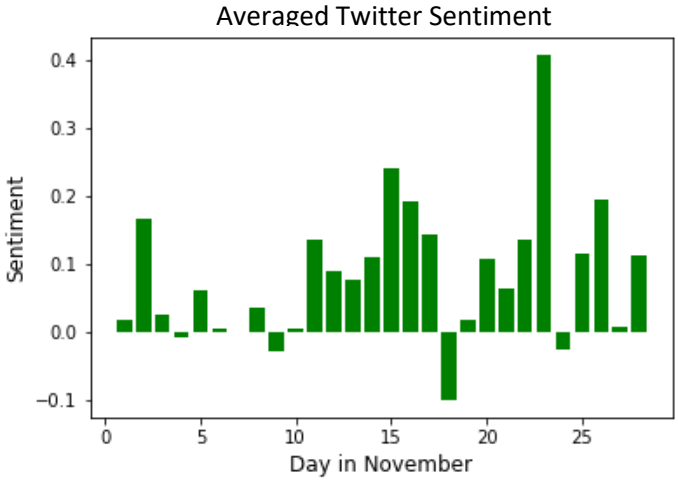


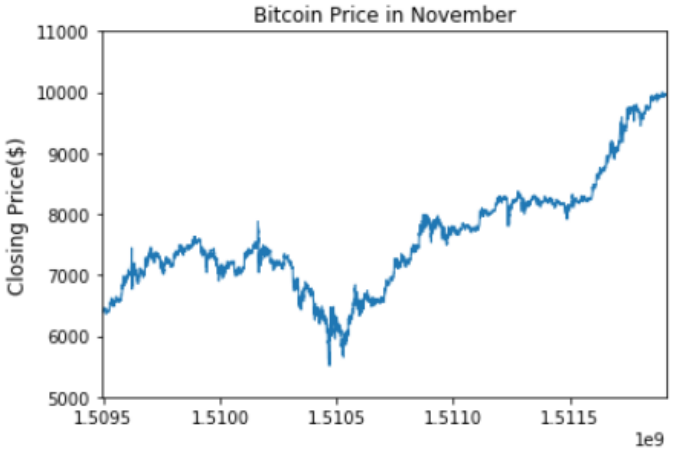
BTC price and see that google searches very closely match BTC price trends. We see that google searches don't give predictive power as they are mostly reactive and don't give additional information.

Next, we look to twitter sentiment to see if that can improve our validation accuracy. We ran into many difficulties with twitter scraping since the official twitter API, Tweepy, only allows tweets from the last 6-9 days to be scraped. A few options exist to get past the limitations Twitter places, but they are cumbersome to use and have unpredictable outputs. The output from these workarounds is also inconsistent, so preprocessing needs to be done to pull out the text from each of the tweets. The best one we found was a scraper from Jefferson Henrique[5].

Because of the difficulties associated with scraping a consistent number of tweets over a long time we opted to do a small case study of a single month. We analyzed November 2017, which was a period of historic growth for Bitcoin where it raised in price from $6000 to $10,000 over one month. Using the VADER sentiment analysis tool[5], we took 30 random tweets from each day, got a compound sentiment score for each tweet and then averaged the score which is given in Figure 10. Because we only took 30 random tweets, we see that the twitter sentiment does not follow the bitcoin price well. When we add this data as an additional input into our model, we do not see any improved performance in accuracy.







Conclusions:

In conclusion, the volatility of the market made our dataset extremely difficult to work with. Future work would have included acquiring more social media text but based on the other poster presentations that specialized in twitter sentiment, this would not improve our results by much. Ultimately, this is a difficult problem to solve because any predictive purchases or sells made by the algorithm would change the market and require the model to account for other neural networks making trades.

Contributions:

Scott Keene: Pricing dataset acquisition, dataset structuring, Keras Model, Google Search
Abdul Obaid: Hyperparameter tuning, Twitter scraping, Data analysis and write-up,
Dante Zakhidov: Baseline Modeling, Vader Sentiment Analysis, Data analysis and write-up

References:

1: Estrada, J., Analyzing Bitcoin Price Volatility. University of Berkeley Doctoral Thesis. (2017)
2: https://www.kaggle.com/mczielinski/bitcoin-historical-data
3. https://trends.google.com/trends/explore?q=bitcoin
4. Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.
5. https://github.com/Jefferson-Henrique/GetOldTweets-python.