Cryptocurrency Price Indicator

04.25.2018

Praveen Origanti

Udacity MLND Capstone Project

I. Definition

Project Overview

Cryptocurrency (or digital currency) is a revolutionary concept enabling peer-to-peer currency exchange without relying on a central trusted authority (like banks). Most cryptocurrencies use a decentralized/distributed ledger system, operating on the principles of cryptography, crowdsourcing and game theory. Satoshi Nakamoto first proposed these principles and defined the first blockchain database whose unit of currency is defined as bitcoin [1].

Bitcoin was created in 2009 and has mostly been limited to exploration by technological enthusiasts. The U.S. Treasury classified bitcoin as a convertible decentralized virtual currency in 2013 and the Commodity Futures Trading Commission, CFTC, classified bitcoin as a commodity in September 2015. Since then, it has garnered greater attention from general public and media. Mostly driven by speculative investments, it has seen an exponential rise in value in the last couple of years and several scholars expect the trend to continue, while others consider it to a bubble. Also, numerous other cryptocurrencies have been created as either variants of bitcoin or new platforms to support other applications. Ethereum, Litecoin and Ripple are a few among the popular ones, also referred to as alt-coins.

Most of the cryptocurrencies posted large gains over the last few years, and peaked in value in December 2017 (as of April 2018). This led to a strong interest in predicting the cryptocurrencies in both financial and academic circles.

Personally, I've been impressed with the technology for a while, but haven't made any investments. I would like to take this opportunity to understand the market (and technology) better and decide if it is a good time to invest in cryptocurrencies.

Problem Statement

The objective is to predict daily percentage returns and daily volatility of the two most popular coins - bitcoin and ethereum - on a given day using the past week's historical market data and appropriate google search trends. These are regression problems, as the goal is to predict continuous numerical values (both returns and volatility).

Since this is a time series problem, I've used recurrent neural networks (RNN), particularly LSTM (Long short-term memory) model, to predict a given day's market behavior based on historical data. I've used Python3 and Keras package with Tensorflow backend.

Metrics

R^2 metric, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are potential metrics for this problem. I've considered both MAE and MSE for the problems of predicting volatility and daily returns. To impose strong penalties on large errors and build aggressive models, I've chosen MSE over MAE as the primary metric for this project.

The model results would be considered satisfactory if the predicted results fare better than benchmark models.

II. Analysis & Methodology

Data Exploration

The historic price data of cryptocurrencies (along with other alt-coins) is available on coinmarketcap.com [3]. For each day from Aug 07 2015 to April 18 2018, I retrieved data from [3] for bitcoin, ethereum, litecoin and ripple that includes open/close/low/high prices, trading volume and market cap. This makes up 986 records of data for all four coins.

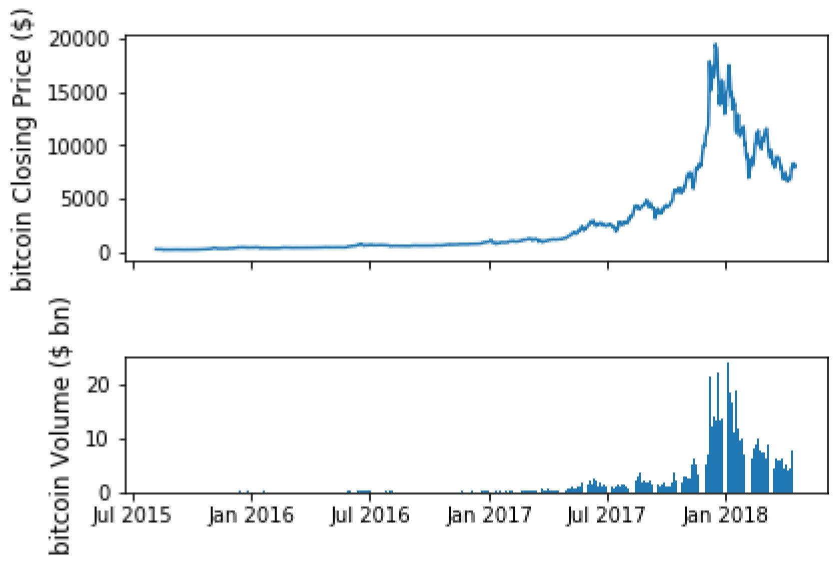


Image 1: Bitcoin’s closing price and trading volume on full dataset

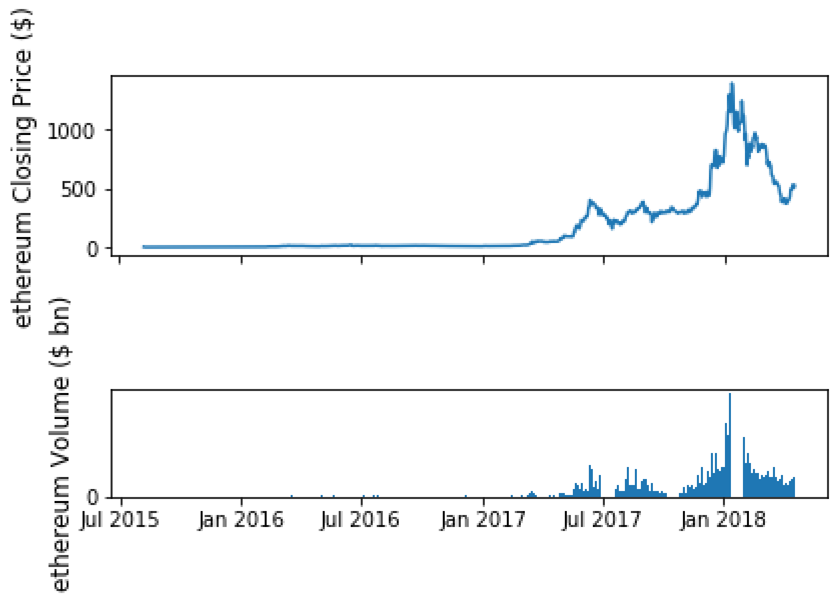


Image 2: Ethereum’s closing price and trading volume on full dataset

As we see in Image 3, there is a strong correlation between Market Cap, Open, Close, High and Low prices of bitcoin. So, I’ve dropped Market Cap, Open, High and Low prices and selected these features - Close price and Volume.

I’ve included these new features based on available market data:

1. Daily returns: (Close - Open)/Open

2. Volatility: (High - Low)/Open

3. Close Off High: (High - Close)/(High - Low)

4. 7-day mean returns: 7-day average returns

5. 30-day mean returns: 30-day average returns

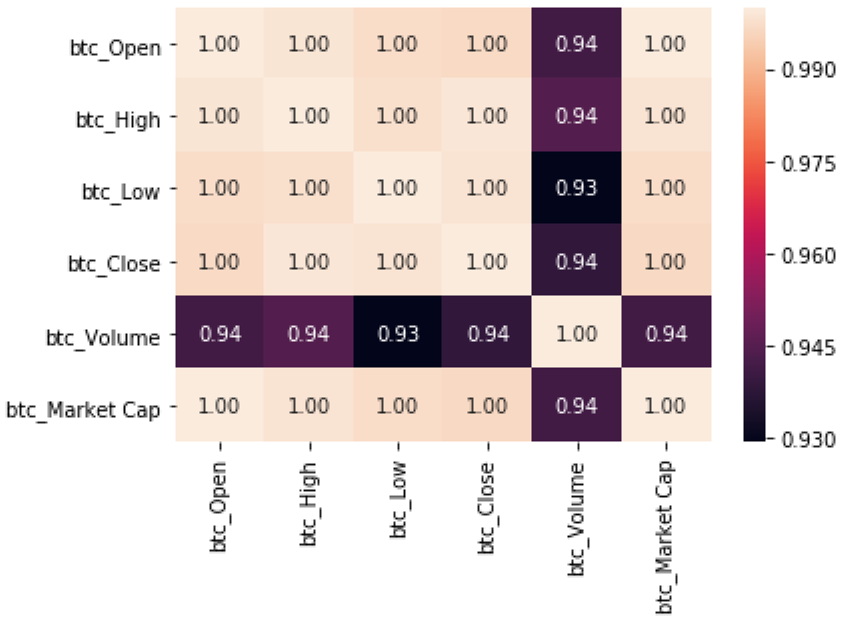


Image 3: Pearson Correlation for Bitcoin’s market attributes

Using the python package pytrends [4], I’ve retrieved the search trends for these four terms - bitcoin, ethereum, litecoin and ripple. For the long duration of interest, Google Trends provided normalized data on a weekly basis. Instead, I’ve generated multiple queries on smaller time durations to retrieve daily data and merged these results by rescaling based on ten overlapped dates in successive queries. Both coinmarketcap and Google Trends use UTC timezone, so there are no issues in merging the datasets.

I’ve used the most recent ~10% as test set (89 records) and the rest as training set (897 records).

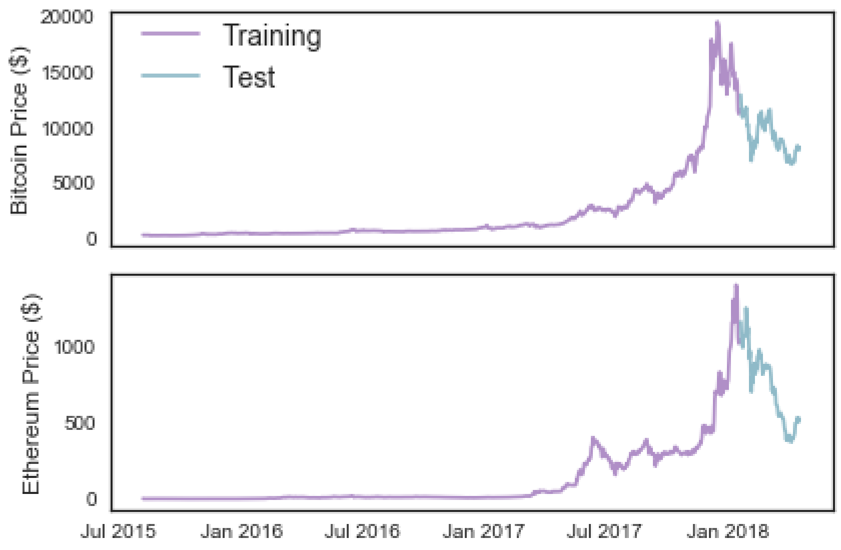


Image 4: Training and test data for Bitcoin and Ethereum (Close Price)

NOTE: Given the high volatility in these markets, it is hard to arrive at useful models just based on price data [2]. In this light, I also wanted to explore the possibility of including twitter/reddit sentiment over this period of time. However, I couldn’t obtain freely accessible twitter/reddit/stocktwits data over this period of time. I could generate the required sentiment data on Quantopian platform [5], but Quantopian has strict limitations on using this data outside their platform. I failed to obtain permissions to use the data for this project. So, I’ve instead used Google Trends data on these search terms - bitcoin, ethereum, litecoin and ripple.

Target Data Exploration:

Instead of predicting the Close Price, I’ve looked at predicting the “Daily returns” of bitcoin and ethereum (Image 5), for the following reasons

- What matters most in any markets is the percentage returns and not the exact price of the commodity/stock/coin.
- We are mostly interested in the returns and not the Close Price
- Daily returns has a naturally gaussian distribution (Image 7)

In addition, I've also predicted "Daily Volatility" of bitcoin and ethereum (Image 6, 8) to see if there are stronger signals here.

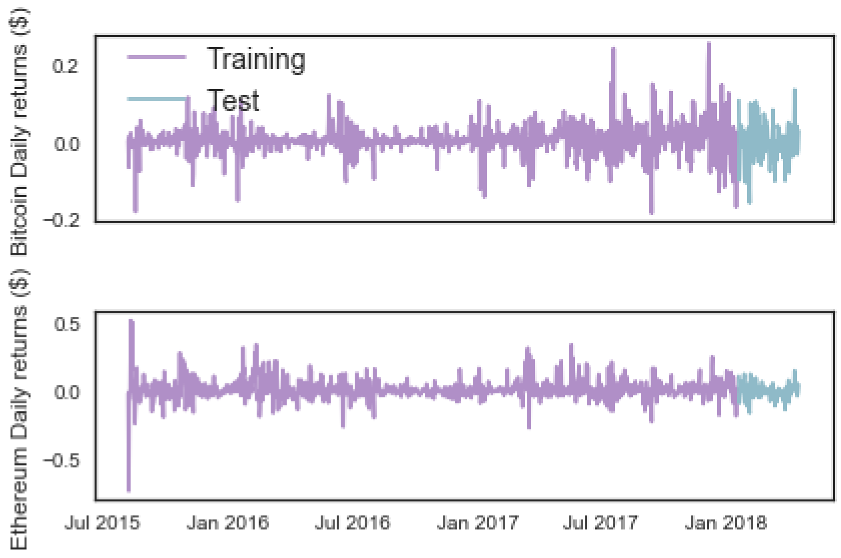


Image 5: Training and test data for Bitcoin and Ethereum (Daily returns)

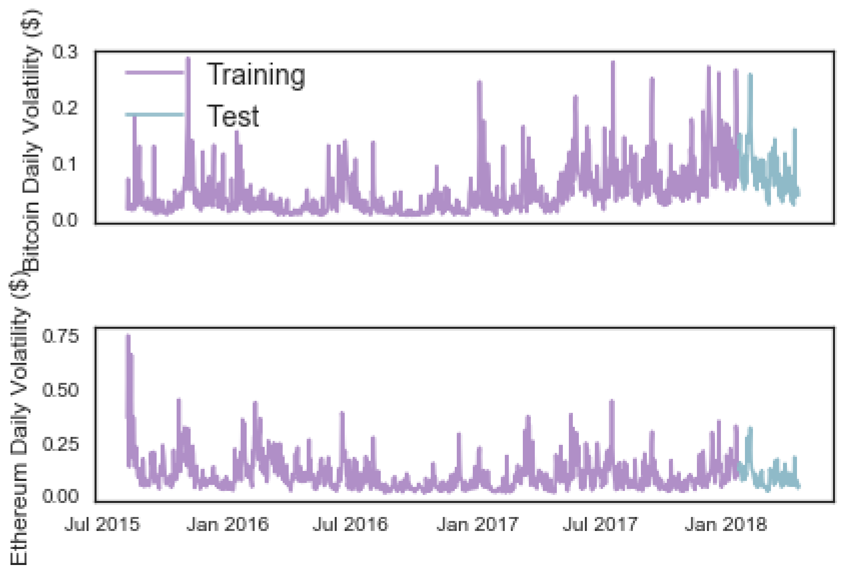


Image 6: Training and test data for Bitcoin and Ethereum (Volatility)

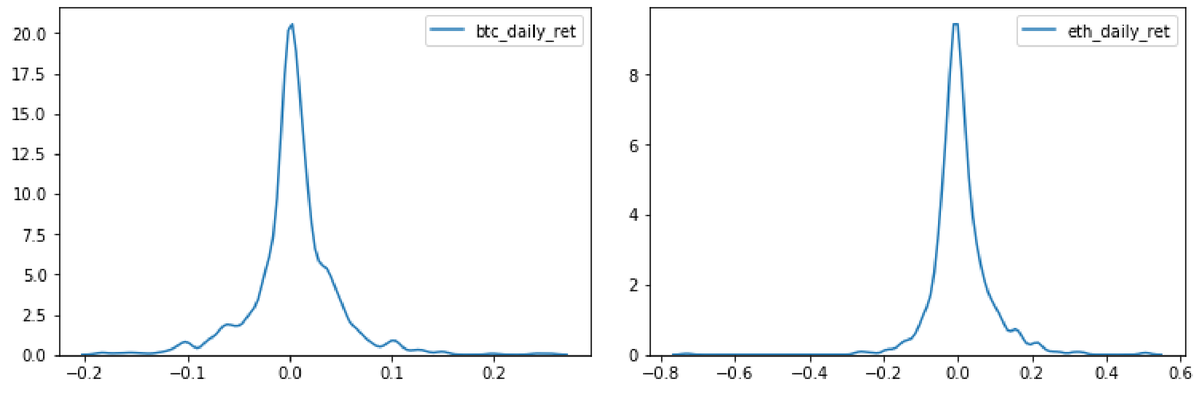


Image 7: Distribution of daily returns for Bitcoin (Left) and Ethereum (Right)

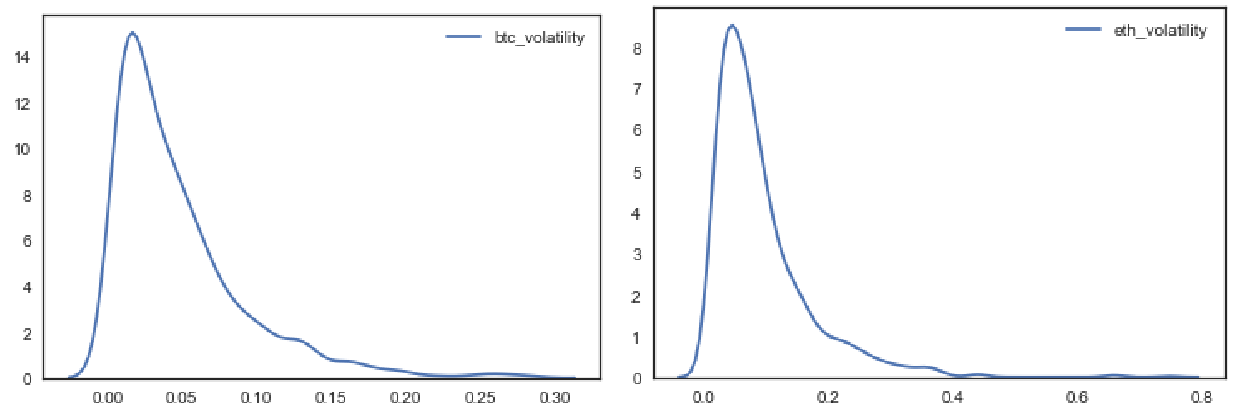


Image 8: Distribution of daily volatility for Bitcoin (Left) and Ethereum (Right)

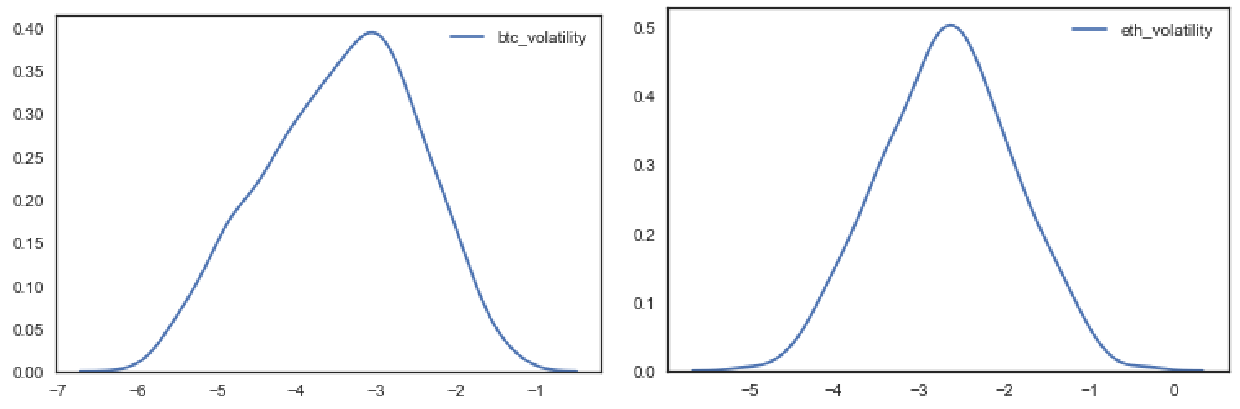
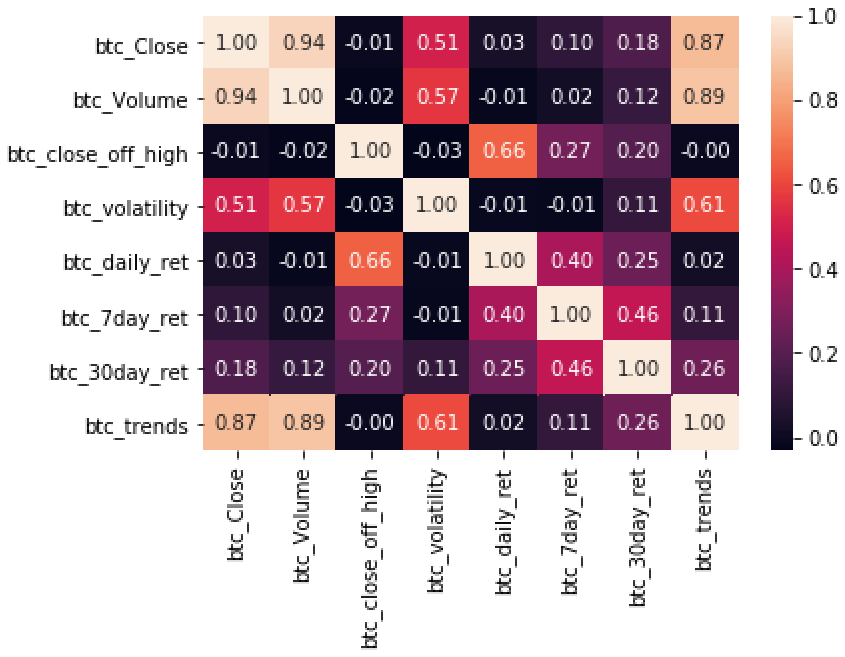


Image 9: Distribution of log(daily volatility) for Bitcoin (Left) and Ethereum (Right)

Given the long tail distributions of “Daily Volatility” of bitcoin and ethereum (Image 8), I’ve applied log transformation and observed closer to gaussian distributions (Image 9).

I’ve appended google search trends to the above set of features. Image 10 captures the pearson correlation of the bitcoin related features used:



	btc_Close	btc_Volume	btc_close_off_high	btc_volatility	btc_daily_ret	btc_7day_ret	btc_30day_ret	btc_trends
btc_Close	1.00	0.94	-0.01	0.51	0.03	0.10	0.18	0.87
btc_Volume	0.94	1.00	-0.02	0.57	-0.01	0.02	0.12	0.89
btc_close_off_high	-0.01	-0.02	1.00	-0.03	0.66	0.27	0.20	-0.00
btc_volatility	0.51	0.57	-0.03	1.00	-0.01	-0.01	0.11	0.61
btc_daily_ret	0.03	-0.01	0.66	-0.01	1.00	0.40	0.25	0.02
btc_7day_ret	0.10	0.02	0.27	-0.01	0.40	1.00	0.46	0.11
btc_30day_ret	0.18	0.12	0.20	0.11	0.25	0.46	1.00	0.26
btc_trends	0.87	0.89	-0.00	0.61	0.02	0.11	0.26	1.00

Image 10: Pearson Correlation for Bitcoin features used.

It can be seen that the search trends for bitcoin (btc_trends) strongly correlates to Close Price (btc_Close), but not much with daily returns. It is interesting to note that there is a decent correlation between btc_trends and btc_volatility (0.61)

Algorithms and Techniques

The objective is to predict the daily percentage returns and volatility of the two most popular coins - bitcoin and ethereum - on a given day using the past week’s (window size = 7) market data and google search trends.

Unlike feedforward neural networks, Recurrent Neural Networks (RNN) have loops in the network and are capable of storing the temporal state. They are best-suited for sequence based learning like speech recognition, text-to-speech translation etc.

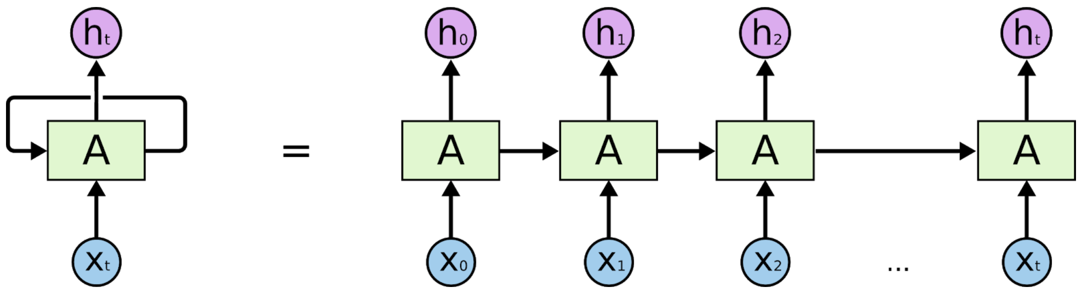


Image 10.1: RNNs have loops. An unrolled RNN has copies of same network, with connections to successors[10]

Long Short-Term Memory (LSTM) units are a popular building block for layers in RNN. LSTM's architecture has an input gate, an output gate and a forget gate (Image 10.2), enabling the networks to remember or forget the internal state as necessary. This lets LSTMs learn temporal correlations in arbitrary time intervals. While training with backpropagation through time, LSTMs are immune to the vanishing or exploding gradient descent[9]

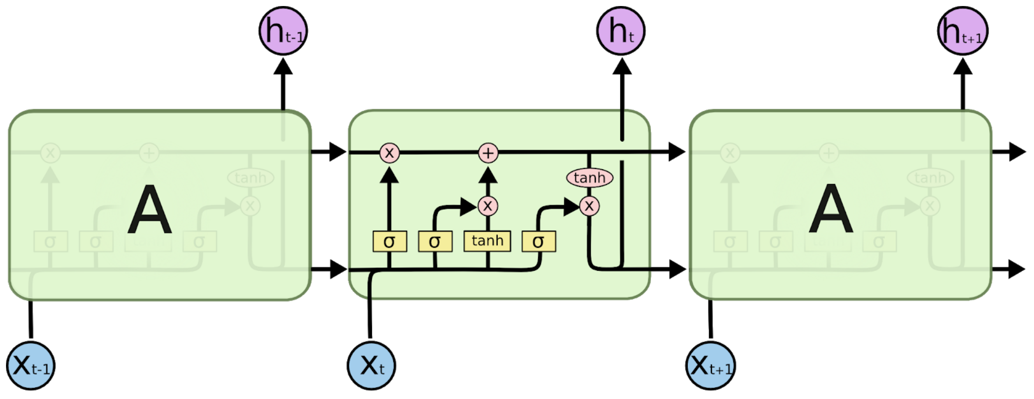


Image 10.2: LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**

I've trained LSTM (Long short-term memory) models to predict the next day's market behavior using historical data. I've explored different combinations of network parameters to determine the final model:

1. Number of stacked LSTM layers
2. Number of neurons in each layer
3. Activation function
4. Dropout
5. Window Size

Benchmarks

I have two benchmarks for each the four prediction problems:

1. Price-persistent model: This assumes the current day's price would stay the same as previous day's close price. In this model, both daily returns and volatility would be zero. This model is a good reference for predicting daily returns
2. Market-persistent model: This assumes the market would behave the same as previous day. In this model, both daily returns and volatility would be same as the previous day. This model is a good reference for predicting daily volatility

Implementation:

The implementation can be split into two primary steps:

1. Data preparation and analysis ('Data Preparation, EDA & Feat Eng' notebook)
2. Benchmarks and training ('Normalization, Benchmarks and training' notebook)

Step #1 includes:

1. Download market data for four coins - bitcoin, ethereum, litecoin and ripple - from coinmarketcap.com using Pandas package from Aug 7, 2015 to April 18, 2018. The data structure used is Pandas DataFrame, and is efficient for pre-processing
2. Obtain google search trends data for the terms - bitcoin, ethereum, litecoin and ripple - using Pytrends package for the same period.
    a. Created helper function get_trends_data() to accumulate the daily trends data by using multiple queries and appropriately rescaling. An interesting exercise to understand and overcome the restrictions on Google Trends.
3. Analyze the dataset by plotting the different features across time. There are no missing values in this data as all four coins are in existence during this time
4. Study/plot the correlations between different features using Seaborn package
5. Select features that are not highly correlated with each other and engineer new features which have the predictive power for this problem - daily returns, volatility, close off high, 7 day and 30 day mean returns. This is achieved using Pandas
6. Analyze the updated dataset and study correlations between the features
7. Log-transform *volatility feature as these features have a long-tail distribution

8. Save this data to pickle format using Pandas. This will make iterations on further processing easier.

Step #2 includes:

1. Read the data from pickle file using Pandas
2. Split dataset into train and test, with the split date being Jan 20, 2018. About 10% of the data is test-set, which constitutes data starting from Jan 20, 2018.
3. Evaluate the two benchmarks on the test set. Get MSE score for the benchmarks
4. To prepare for LSTM, specify window size to 7. In the later stages, I've tried these values for window size: 1, 2, 7, 14, 30
5. Create training and test inputs for LSTM by creating sliding windows of all features with each window consisting of 7-day worth data
   a. Due to the way the windows are created, the target from initial 7-days of training set cannot be used. However, by using the last 7-days of training set, we can take advantage of the entire test set. I did take a while to identify and fix a minor bug in implementing this correctly.
6. Create LSTM model using Keras package. The network architecture consists of 3 stacked layers of LSTM and the final dense layer (activation function: linear/tanh) with one output (Image 11). Each LSTM layer is followed by dropout, to generalize better.
7. Normalize features - Close Price, Volume and Trends - in reference to first day of the window. More details on the normalization approach below.
8. Split training data into 5 folds using Scikit-learn's TimeSeriesSplit. This ensures that no validation data occurs prior to any training data during cross-validation
9. Explore the parameters space using Scikit-learn's GridSearchCV and find the best set of parameters for each prediction problem. Later sections have details
   a. One challenge I faced is to figure out about KerasRegressor wrapper which is part of keras.wrappers.scikit_learn. This helped with seamless integration of Keras model with Scikit-learn workflow
10. Analyze/visualize the results and compare performance with results from benchmark models.
11. Perform sensitivity analysis on the final best model by adding random noise to the train and test dataset (ten times), and statistically analyzing the ten results.

Normalization

For the time series, I've followed the normalization approach described here [6] - for each sliding window, scale the values using the first record in the window as the reference such that the first record's value would be zero (after scaling)

I've used the following equations to normalise:

n = normalised list [window] of price changes

p = raw list [window] of adjusted daily return prices

Normalisation:

$n_i = (p_i/p_0)-1$

De-Normalisation:

$p_i = p_0(1+n_i)$

The Close Price, Volume and Google Trends Data are normalised this way. I didn't have to de-normalise since these are independent features in our prediction models.

GridSearch & Cross-Validation

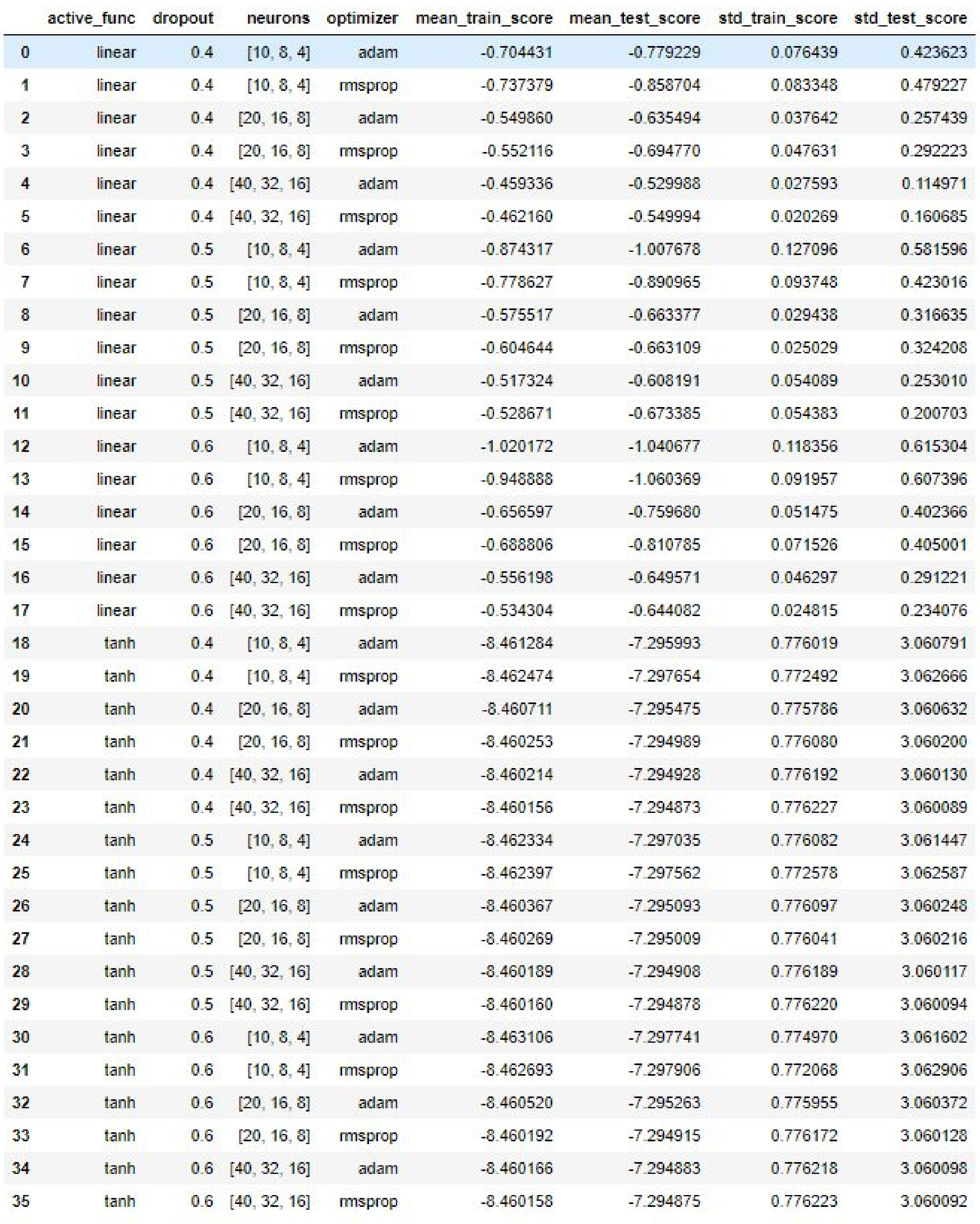
Using sklearn's TimeSeriesSplit, five CV folds are created - successive training sets would be supersets of the previous ones. This ensures that the folds are not created randomly - and the principles of time series data are strictly adhered to.

For bitcoin volatility prediction, the details of the explored hyperparameters and the corresponding results are captured in Image 10.3

Refinement

Some of the refinements I've made during the process:

1. Switching to Mean Squared Error (instead of Mean Absolute Error) to build aggressive models
2. Obtained additional relevant data - Litecoin and Ripple market & trends data
3. Include the hyperparameter 'optimizer' for optimization using GridSearchCV
4. Explored with multiple window sizes, outside of GridSearchCV
5. Predicted market volatility in addition to daily returns, to explore further.



	active_func	dropout	neurons	optimizer	mean_train_score	mean_test_score	std_train_score	std_test_score
0	linear	0.4	[10, 8, 4]	adam	-0.704431	-0.779229	0.076439	0.423623
1	linear	0.4	[10, 8, 4]	rmsprop	-0.737379	-0.858704	0.083348	0.479227
2	linear	0.4	[20, 16, 8]	adam	-0.549860	-0.635494	0.037642	0.257439
3	linear	0.4	[20, 16, 8]	rmsprop	-0.552116	-0.694770	0.047631	0.292223
4	linear	0.4	[40, 32, 16]	adam	-0.459336	-0.529988	0.027593	0.114971
5	linear	0.4	[40, 32, 16]	rmsprop	-0.462160	-0.549994	0.020269	0.160685
6	linear	0.5	[10, 8, 4]	adam	-0.874317	-1.007678	0.127096	0.581596
7	linear	0.5	[10, 8, 4]	rmsprop	-0.778627	-0.890965	0.093748	0.423016
8	linear	0.5	[20, 16, 8]	adam	-0.575517	-0.663377	0.029438	0.316635
9	linear	0.5	[20, 16, 8]	rmsprop	-0.604644	-0.663109	0.025029	0.324208
10	linear	0.5	[40, 32, 16]	adam	-0.517324	-0.608191	0.054089	0.253010
11	linear	0.5	[40, 32, 16]	rmsprop	-0.528671	-0.673385	0.054383	0.200703
12	linear	0.6	[10, 8, 4]	adam	-1.020172	-1.040677	0.118356	0.615304
13	linear	0.6	[10, 8, 4]	rmsprop	-0.948888	-1.060369	0.091957	0.607396
14	linear	0.6	[20, 16, 8]	adam	-0.656597	-0.759680	0.051475	0.402366
15	linear	0.6	[20, 16, 8]	rmsprop	-0.688806	-0.810785	0.071526	0.405001
16	linear	0.6	[40, 32, 16]	adam	-0.556198	-0.649571	0.046297	0.291221
17	linear	0.6	[40, 32, 16]	rmsprop	-0.534304	-0.644082	0.024815	0.234076
18	tanh	0.4	[10, 8, 4]	adam	-8.461284	-7.295993	0.776019	3.060791
19	tanh	0.4	[10, 8, 4]	rmsprop	-8.462474	-7.297654	0.772492	3.062666
20	tanh	0.4	[20, 16, 8]	adam	-8.460711	-7.295475	0.775786	3.060632
21	tanh	0.4	[20, 16, 8]	rmsprop	-8.460253	-7.294989	0.776080	3.060200
22	tanh	0.4	[40, 32, 16]	adam	-8.460214	-7.294928	0.776192	3.060130
23	tanh	0.4	[40, 32, 16]	rmsprop	-8.460156	-7.294873	0.776227	3.060089
24	tanh	0.5	[10, 8, 4]	adam	-8.462334	-7.297035	0.776082	3.061447
25	tanh	0.5	[10, 8, 4]	rmsprop	-8.462397	-7.297562	0.772578	3.062587
26	tanh	0.5	[20, 16, 8]	adam	-8.460367	-7.295093	0.776097	3.060248
27	tanh	0.5	[20, 16, 8]	rmsprop	-8.460269	-7.295009	0.776041	3.060216
28	tanh	0.5	[40, 32, 16]	adam	-8.460189	-7.294908	0.776189	3.060117
29	tanh	0.5	[40, 32, 16]	rmsprop	-8.460160	-7.294878	0.776220	3.060094
30	tanh	0.6	[10, 8, 4]	adam	-8.463106	-7.297741	0.774970	3.061602
31	tanh	0.6	[10, 8, 4]	rmsprop	-8.462693	-7.297906	0.772068	3.062906
32	tanh	0.6	[20, 16, 8]	adam	-8.460520	-7.295263	0.775955	3.060372
33	tanh	0.6	[20, 16, 8]	rmsprop	-8.460192	-7.294915	0.776172	3.060128
34	tanh	0.6	[40, 32, 16]	adam	-8.460166	-7.294883	0.776218	3.060098
35	tanh	0.6	[40, 32, 16]	rmsprop	-8.460158	-7.294875	0.776223	3.060092

Image 10.3: Hyperparameters explored and corresponding scores (on 5-folds) for bitcoin volatility prediction. NOTE: GridSearchCV negates the MSE score to indicate 'higher-is-better'

III. Results

Model Evaluation and Validation

1. Predicting Daily Returns

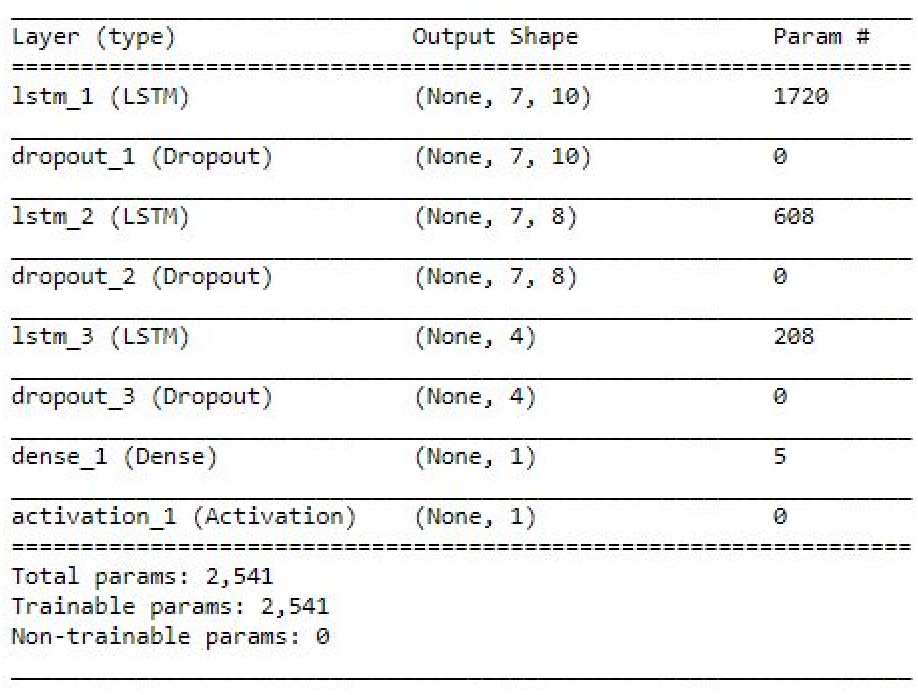
For predicting daily returns of bitcoin and ethereum, based on the GridSearchCV results, the model in Image 11 obtained the best scores. However, the best MSE score was not satisfactory as it is about the same performance as benchmark-I (Table -I)

The predicted returns are observed to be pretty close to 0 most of the time, likely because there are no clear patterns that could be learned from this data.

	BTC Returns	ETH Returns	BTC Volatility	ETH Volatility
Benchmark-I	0.00306	0.00354	7.188	7.075
Benchmark-II	0.006	0.007	0.282	0.266
Best LSTM model	0.00305	0.00362	0.232	0.250

2. Predicting Daily Volatility

For predicting daily volatility of bitcoin/ethereum, a slightly more complex model (neurons: [20, 16, 8]) achieved the best score. The best MSE scores for bitcoin volatility and ethereum volatility are 17% and 6% better their benchmark-II scores (Table-I).



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 7, 10)             1720      
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 10)             0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 7, 8)              608       
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 8)              0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 4)                 208       
_________________________________________________________________
dropout_3 (Dropout)          (None, 4)                 0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 5         
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0         
=================================================================
Total params: 2,541
Trainable params: 2,541
Non-trainable params: 0
_________________________________________________________________
```

Image 11: Final LSTM network used for Bitcoin & Ethereum's Daily Volatility Prediction. Neurons: [10, 8, 4]; Dropout=0.4; Optimizer=adam; activation='linear'

Sensitivity Analysis

To validate the robustness of the final model, I've performed these steps:

1. Inject random noise into all the features of the dataset (train and test). For each feature, add random noise with uniform distribution in the range [-a, a], where a is one-hundredth of the standard deviation of the feature.
2. Predict the performance of the final model on 10 such noise-injected datasets
3. Measure/plot the prediction scores (Image 11.1)
    a. Training set
        i. Original score: 0.5507
        ii. Mean score on noisy train set: 0.5602; Std deviation: 0.0046
    b. Test set:
        i. Original score: 0.2090
        ii. Mean score on noisy test set: 0.20898; Standard deviation: 0.00004

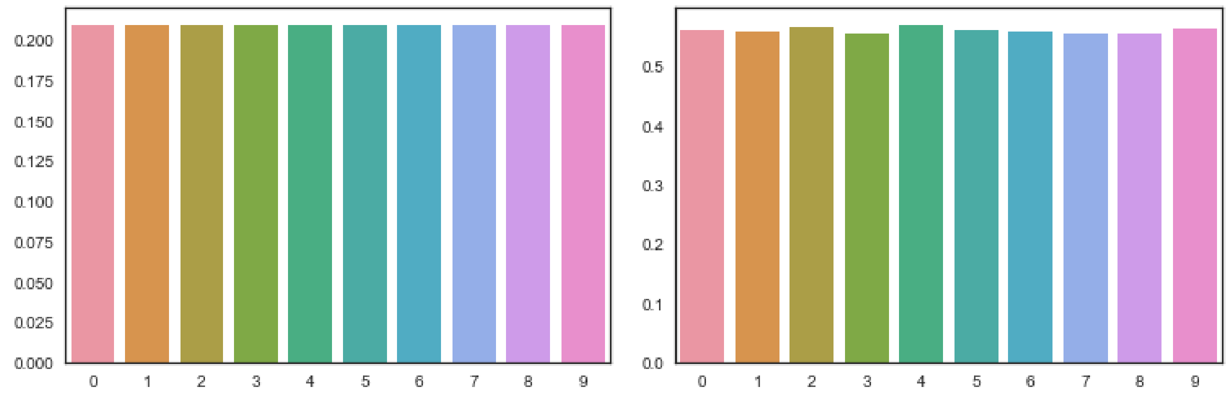


Image 11.1: The final model’s predictions on test (Left) and train (Right) of the ten noise-injected datasets

Justification

The predictions on daily returns are barely useful given that they just match benchmark model-I. Interestingly, the predictions on volatility are better than the best benchmarks indicating that it is worth pursuing in this direction. However, I am not sure if predicting market volatility is useful (for investors, i.e.). One potential use-case is to measure the risk involved in these markets.

I don’t consider the results to be significant enough, but it can be attributed to several aspects - primarily the unavailability of patterns/signal in this data.

IV. Conclusions

Free-Form Visualization

The most interesting observation for me is that LSTM models for daily returns ended up predicting values close to 0, while the actual market returns have huge variations (Image 12). To me, this is a strong indicator that there is little signal in the available data for LSTM model to infer useful information.

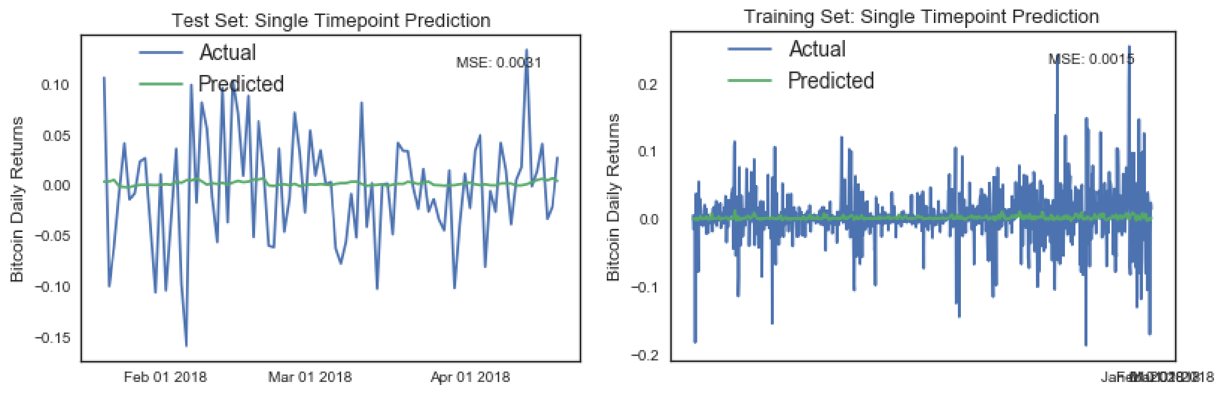


Image 12: Actual & Predicted daily returns of Bitcoin on test (Left) and train (Right) data

On the other hand, the predictions on volatility have a larger variance (still conservative compared to market behavior), indicating a potential signal in this data. (Image 13)

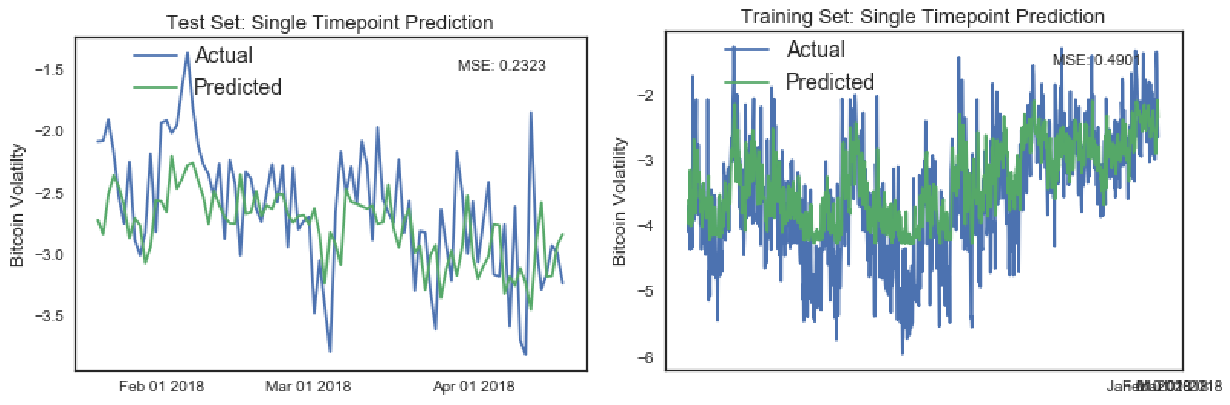


Image 13: Actual & Predicted daily volatility of Bitcoin on test (Left) and train (Right) data

Reflection

In summary, below are the steps used for this project:

- Define the problem statement
- Identify potential datasets useful for the problem

- Obtain data from various sources and preprocess data
- Define benchmarks for the problems
- Define initial LSTM model with one layer, and default parameters
- Explore the model space and optimize the hyperparameters using GridSearch
- Evaluate and visualize the results and performance

I am excited to have applied machine learning for predicting cryptocurrencies. I gained deeper understanding of these two very exciting fields - machine learning and digital currencies. Thanks to excellent libraries and blog posts, I am able to develop this project in such a short span of time (3-4 weeks of effort). I am extremely satisfied and inspired to confidently develop prototypes for some of the other projects I’ve in mind.

Although I hoped to find ways to beat the ‘efficient market hypothesis’ in crypto markets, I’ve learned quickly that this is a complex problem. And even if the model worked, it very likely would work for only a brief period of time. I wouldn’t be surprised if the markets adapt and would behave very differently.

In the process, I’ve also learned about Google Trends, Google Correlate (this is an eye-opener), Google Colab and Quantopian platform for algorithmic trading.

Also, I’m extremely satisfied to have explored this problem so far. I wouldn’t propose or use the project (in its current state) as a guide for investments. In the near future, I plan to import this work into Quantopian platform and leverage the additional sentiment data available in there.

Scope for Improvement

A few aspects that can certainly improve the training process:

- Better data that potentially has higher predictive power (eg. investor sentiment)
- GridSearchCV failed with njobs=-1. The parallelism helps explore better
- Batch Normalization layer, which I haven’t used in these models
- Early Stopping - not integrated with GridSearchCV. But there must be a way

I believe better data (investor sentiment) can help. Based on hyperparameter tuning, more complex LSTM architectures did not provide better results, so I don’t think using complex architectures would help

References

1. https://bitcoin.org/bitcoin.pdf
2. https://dashee87.github.io/deep%20learning/python/predicting-cryptocurrency-prices-with-deep-learning/
3. https://coinmarketcap.com/
4. https://github.com/GeneralMills/pytrends
5. https://www.quantopian.com
6. http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction
7. https://keras.io/layers/recurrent
8. https://stackoverflow.com/questions/48127550/early-stopping-with-keras-and-sklearn-gridsearchcv-cross-validation
9. https://en.wikipedia.org/wiki/Long_short-term_memory