Bidirectional LSTM time series prediction on crypto price [wip]

Kim Aksel Tahuil Borgen

May 13, 2019

Abstract

This report is currently in Work In Progress modus. Some parts are incomplete and a general spellwash is required. But the report is still readable if you are curious.

The purpose of this project was to learn the theory behind Recurrent Neural Networks (RNNs) and to implement a solution. The purpose was not to gain a positive result but rather gain an insight into methods for time series prediction.

In this project I built models to predict if the price of Ethereum will be more or less than the price of the previous day. The models where trained and tested upon the entirity of Ethereum's historical price data. This model was made using Recurrent Neural Networks, which is known to produce good results on time series prediction. More specifically I used a model based upon Bidirectional Long Short Term Memory Recurrent Neural Networks.

I achieved an accuracy of 58.24% with little to none *Feature Engineering* and *Data Augmentation*, which is a bit better than the baseline performance of 55.72% of another classical machine learning method, and much better than a random guess or a naive trading strategy.

Source-code is available upon request.

Table of Contents

1 Data acquisition and initial data discussion

The historical data was collected from the API that coingecko.com provides for free. The data includes daily price, market cap and trading volume for Ethereum. Smaller time segments where not available. This data can be seen plotted in figure 1.

I have also included an additional plot, that can be seen in Figure 2, of the market cap of various other different cryptocurrencies. We can see here that almost all currencies are somewhat correlated. Based on

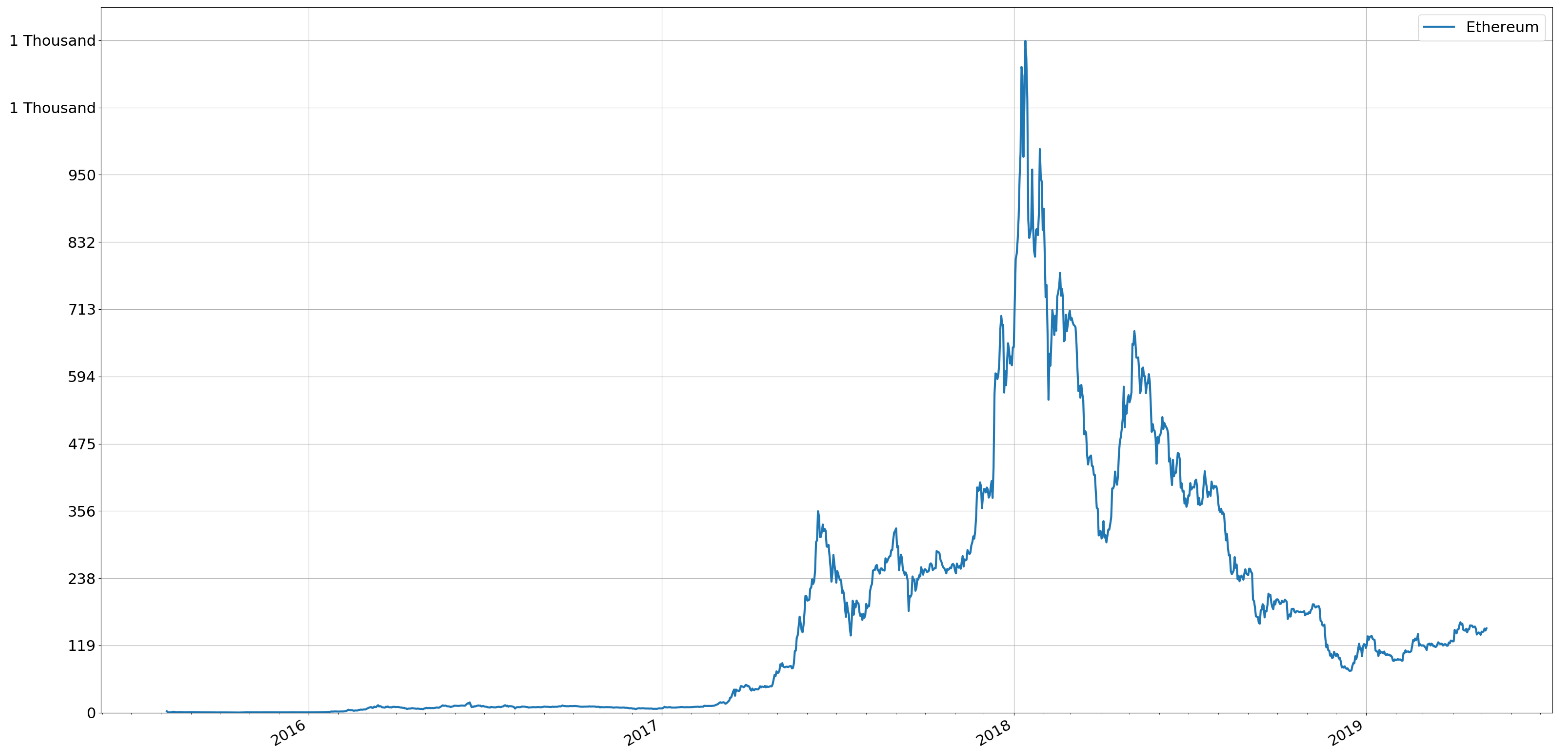


Figure 1: Historical price of Ethereum (ETH)

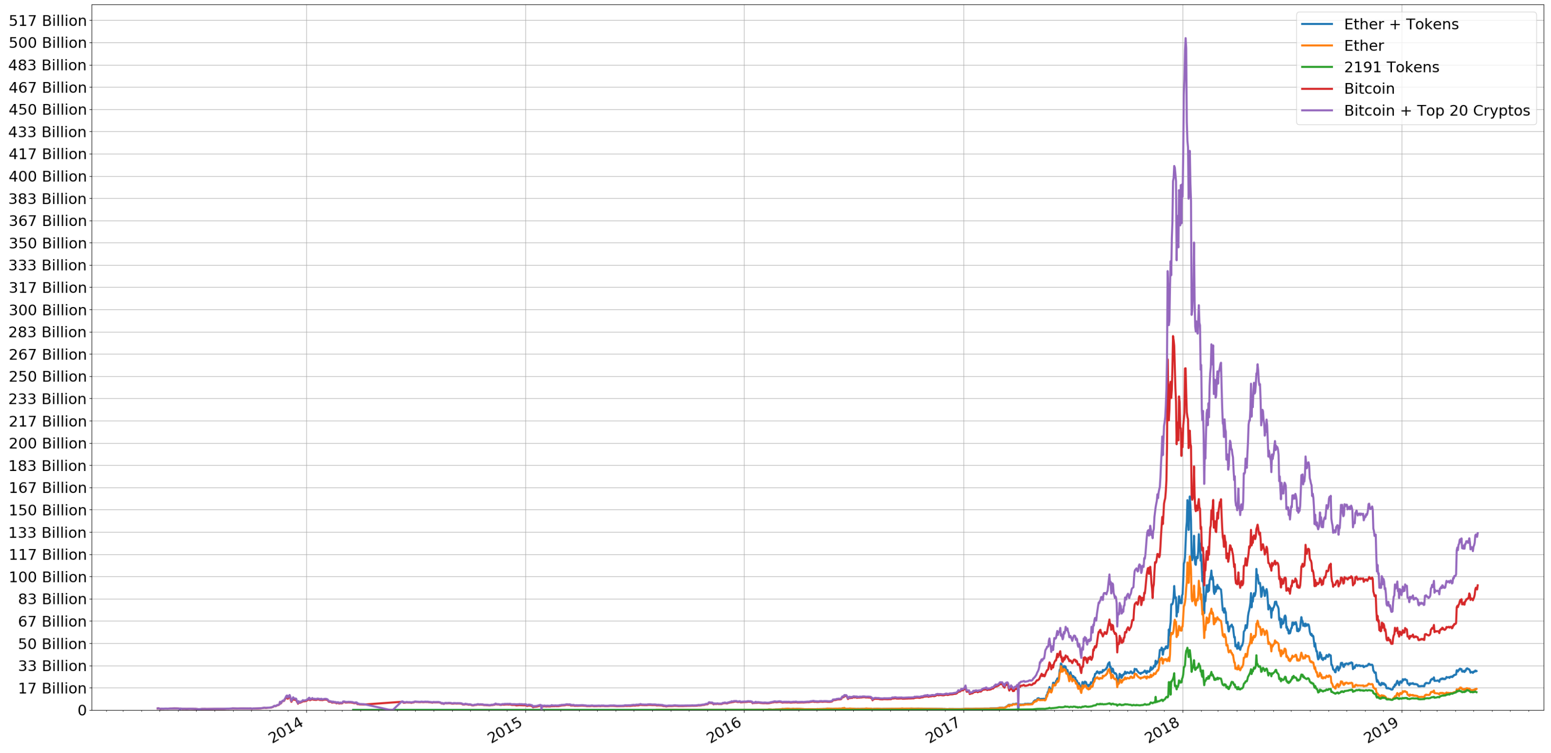


Figure 2: Historical market cap of different cryptocurrencies

this we can produce an hypothesis that a model trained on one currency will perform similar on another currency.

2 Baseline performance

In order to evalute if the neural network performs better than other models, or other trading strategies, a baseline performance must be estahblished.

An early hypothesis was that the Ethereum price, on average, goes up. But upon investigation of the data the amount of days with positive growth in relation to the day before was 679 days, and similary, the amount of down days was 688. Making it almost a 50% percentage of going up or down, the same as choosing randomly. So the naive trading strategy of simply choosing up will not work.

Since any guess above 50% is better, an older, more battle-tested method should be tested in order to produce a baseline. In this scenario I picked Support Vector Machines (SVM) which achived an accuraccy of 55.72%. SVM was chosen purley since I had previous experience and knowledge on SVM's and not because they produce good results on this kind of dataset.

3 Theory

Recurrent Neural Networks (RNN) are excellent for time series data, that is, sequential data that embodies correlations between data points that are close in sequence [4]. RNNs basically works by remembering past states, and uses that to predict future states. Bidirectional RNNs work by also predicting past state from future data and combining the results with normal RNNs [4]. This approach will most likely give better results. Long Short Term Memory (LSTM) is a special RNN architecture that is very general purpose, it is even Turing complete! LSTM is also very efficient at time series predictions. When implementing continuous prediction, LSTM with forget gates will be even more efficient [2], which is very useful in financial time series which is continuous. A somewhat up to date research paper on LSTM can be found in Greff et al's paper from 2017 [3].

4 Method

Preprocessing: Based upon the hyperparamter *moving_window* the dataset was split into seperate lists of size *moving_window* by iterating on the price data and picking the next *moving_window* timesteps for each list, and then doing the same on the next timestep.

Each list was then normalized based on the first elements of the lists by using the following code:

```
for i, window in enumerate(window_list):
    normalized[i] = window_list[i, :] / window_list[i, 0] - 1
```

On the first attempt I split the normalized data 80/20 into a training set and a testing set. Then I randomly shuffled the training set. This produced a testing set containing data of 272 continous days, making it easy to compare the results to a real time period.

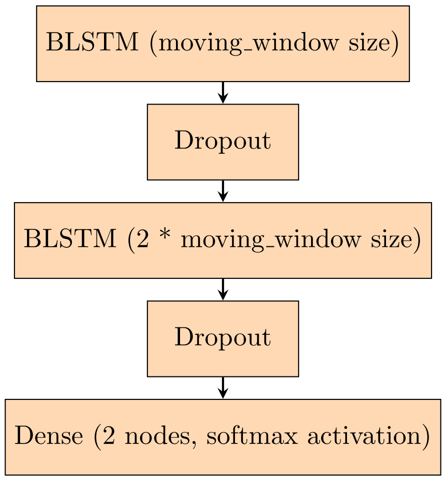
On the second attempt I first shuffeled the data, and then split it into a training and testing set. add result here

Training: We train the network using the *sparse categorical crossentropy* loss function and *adam* optimizer.

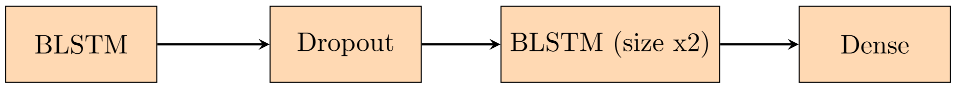
Evaluating: The model is then evaluated using the testing set. The evalutaion method first counts up the number of correct predictions and then divides it by the total length of the testing set giving a percentage score.

5 Tools and network architechture

For the machine learning part I used Keras with a Tensorflow backend, and for the data processing part I used Numpy. The model was compiled based on other models that predict financial time series. Not that much thought was put into the construction of this network since finding the correct model could be a very much more daunting task. The RNN architecture can be seen below. *Note: BLSTM stands for Bidirectional LSTM.*


BLSTM (moving_window size)
Dropout
BLSTM (2 * moving_window size)
Dropout
Dense (2 nodes, softmax activation)

I also compiled a model to do regression, instead of classification, in order to actually predict the price. But because of time constraints this was not tested. The model achived similar results and is therefor included below.



6 Hyperparameter selection

There are many hyperparamters to choose in an RNN architechture, but few on a SVM model.

I optimize hyperparamters by utilizing a random search on the three variables: moving window, batches and ephocs. Other hyperparameters where not optimized because I lack the time and processing power to do a more extensive random search. The random search method has good precendes, and produces better results than manual and grid searching methods [1].

The SVM model was only optimized on the moving window variable by simply looping trough sizes 1 to 500 and picking the best.

7 Results

From the neural networks output softmax layer we get back a prediction in the form of *[percent chance of positive price change, percent chance of negative price change]*, where the sum is 1. It should be noted that the percentages lay in the range (46,54) and some are very close.

7.1 Known testing dataset

The SVM achived an accuracy score of 55.72% using a moving window of 12 days.

The classification LSTM achived an accuracy score of 58.24% with the following hyperparameters: Moving window of 65 days, batches of 209, 14 training ephocs, validation split of 5%, and a dropout rate of 20%.

The test set had 119 positive days and 142 negative days. The model predicted 38 positive days and 223 negative days. The plot of the real and predicted results can be seen in Figure 3. In comparison, the test dataset is plotted in Figure 4

The regression LSTM achived an accuracy of 57.2%, with the following hyperparameters: moving window of 15 days, 71 batches, 12 ephocs, validation split of 5%, and a dropout rate of 20%.

7.2 Random testing dataset

The classification LSTM achived an accuracy score of 57.88% with the following hyperparameters: Moving window of 5 days, batches of 415, 11 training ephocs, validation split of 5%, and a dropout rate of 20%.

The program is still searching for the optimal hyperparamters, I will update this section when I am satisfied with the search.

8 Discussion and future work

The result of 58.24% accuracy is pretty good in theory. With the 272 days in the testing set, this algorithm is correct 21 days more than a 50% guess and 6 days more than the SVM.

One interesting result from the predicted values on the known testing dataset is that the model mainly predicted negative price days. But as we can see in Figure 4, the period was mostly negative and therefor classified as a bear period.

From the results I can conclude with an hypothesis that this neural network can be used to correctly predict values in well defined periods, but may not produce good results when predicting macro changes, such as moving from a bear market to a bull market.

This hypothesis can be tested by identifying such changes and produce a new testing set based on this, but this is outside the scope of this projet and will therefor be labeled future work.

A better result can maybe be achieved with more *Feature Engineering*, that is, to find or extract additional features to expand the feature set. *Data Augmentation*, that is, to generate additional data or noise to expand the dataset, could also prove beneficial, especially since the dataset of only one currency is relatively sparse.

The simplest way to expand the feature set would be to simply include historical price data from other cryptocurrencies as well.

But more important features could for example be other statistics, such as semantic analysis, blockchain network statistics (such as number of transactions, transaction delay, number of dapps, ...) and even an analysis on how the general stock market performs.

References

[1] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[2] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM" (1999).

[3] Klaus Greff et al. "LSTM: A search space odyssey". *IEEE transactions on neural networks and learning systems* 28.10 (2017), pp. 2222–2232.

[4] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

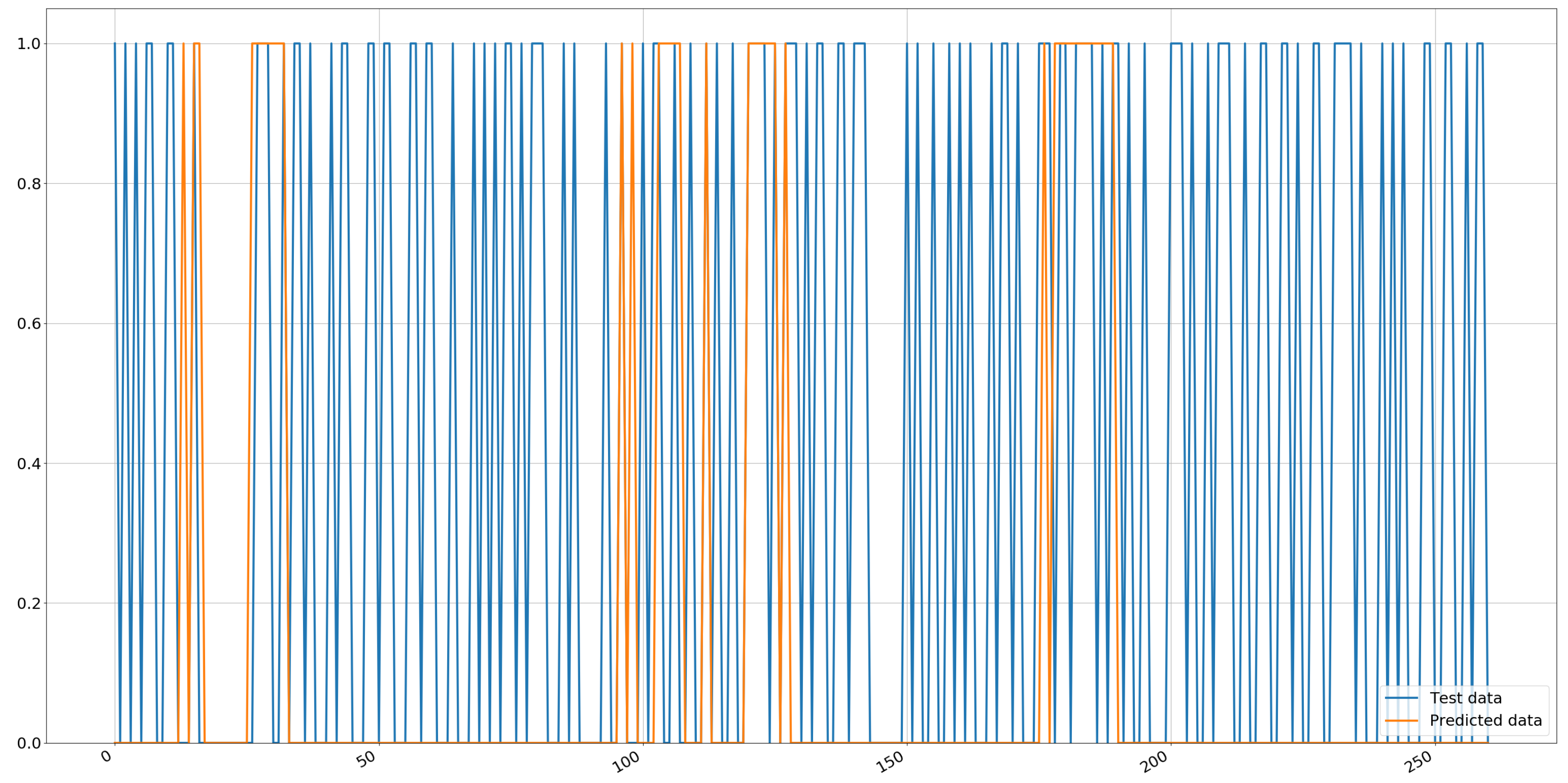


Figure 3: Plot of real and predicted positive/negative days

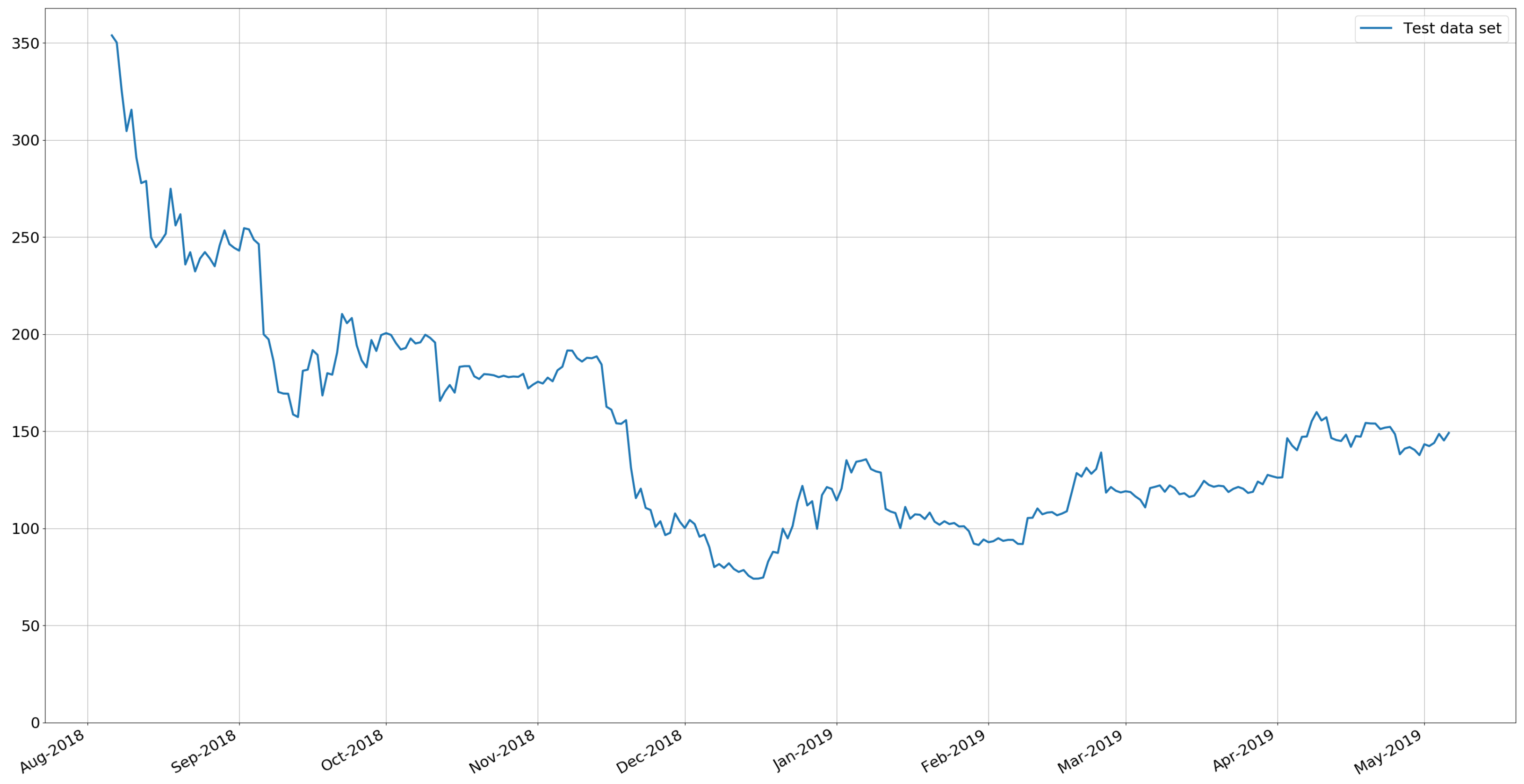


Figure 4: Test dataset