Machine Learning Engineer Nanodegree Capstone Report

Daniel Chao Zhou

31 December 2019

1 Definition

1.1 Project Overview

The stock market prediction has been identified as a significant practical problem in the economic field. Trading algorithms rather than humans performed over 80% of trading in the stock market and the FOREX market. In the crypto-currency market, algorithmic trading is also a hot topic among investors. However, timely and accurate prediction of the market is generally regarded as one of the most challenging problems, since the environment is profoundly affected by volatile political-economic factors, such as legislative acts, political unrest, and mass irrational panic.

There are many studies regarding algorithmic trading in financial markets based on machine learning, where recurrent neural network (RNN) and reinforcement learning (RL) are being popular in recent years. In this study, a Bitcoin price predictor based on long short-term memory (LSTM, a variant of RNN) is presented.

1.2 Problem Statement

Given the time-series trading data of a Bitcoin futures contract with each time step indicating one minute, the goal is to build a predictor for the volume-weighted average price (VWAP) of the next minute.

In this study, a price predictor based on LSTM will be built.

1.3 Metrics

The root-mean-square deviation (RMSD) between labels y and predictions $\hat{y}$ will be used to evaluate the performance of both of the benchmark model and the solution model. RMSD is defined as follows,

$$\begin{aligned} \mathrm{MSE}(y, \hat{y}) &:= \frac{1}{N} \sum_{i} (y_i - \hat{y}_i)^2, \\ \mathrm{RMSD}(y, \hat{y}) &:= \sqrt{\mathrm{MSE}(y, \hat{y})}. \end{aligned}$$

For each time step t, the label y_t is defined as the VWAP of the next time step,

$$y_t := \texttt{vwap}_{t+1}.$$

2 Analysis

2.1 Data Exploration

In this study, trading data of BitMEX's XBTZ19 contract, which is a Bitcoin futures contract being traded around the clock from 16 June to 27 December 2019, will be used to train and test the predictor. The dataset could be fetched from BitMEX's official API without charge of fee.[1]

The dataset is a table that each row indicates one minute and each column indicates a specific data as described in Table 1. Note that `open` is not defined as the price of the first trade in the specific time step, which is an unconventional definition and does not apply to other data sources.

The dataset is formulated as $\{x_t | t = 1, 2, \ldots, T\}$, where x_t is a vector of the data in minute t, such that $x_t = (\texttt{open}_t, \texttt{high}_t, \texttt{low}_t, \texttt{close}_t, \texttt{vwap}_t, \ldots)$. A small sample and some basic statistics are shown in Table 2 and Table 3.

[1]The API concerned with the desired data is documented at `https://www.bitmex.com/api/explorer/#!/Trade/Trade_getBucketed`

`high`	highest price
`low`	lowest price
`close`	price of the last trade
`open`	`close` of the last time step
`vwap`	volume-weighted average price (VWAP)
`foreignNotional`	traded amount in units of US dollar
`homeNotional`	traded amount in units of Bitcoin
`trades`	number of trades
`volume`	alias of `foreignNotional`

Table 1: Column header semantics

	open	high	low	close	vwap	foreignNotional	homeNotional	trades
2019–06–14 08:31	NaN	NaN	NaN	NaN	NaN	0	0.000000	0
2019–06–14 08:32	NaN	NaN	NaN	NaN	NaN	0	0.000000	0
2019–06–14 08:33	8500.00	8500.00	8260.00	8260.00	8262.4143	201	0.024328	3
2019–06–14 08:34	8260.00	8390.00	8308.00	8308.00	8325.0083	13110	1.574852	7
2019–06–14 08:35	8308.00	8390.00	8319.50	8336.50	8322.9297	13200	1.585986	5
2019–06–14 08:36	8336.50	8317.50	8317.50	8317.50	8317.5000	10200	1.226346	3
2019–06–14 08:37	8317.50	8327.50	8327.50	8327.50	8328.0000	500	0.060040	1
2019–06–14 08:38	8327.50	8366.50	8362.50	8362.50	8365.4007	4001	0.478290	5
...	...	...	...	...	...	...	...	...
2019–12–27 11:54	7151.00	7155.50	7135.00	7137.50	7149.4960	632224	88.434680	62
2019–12–27 11:55	7137.50	7149.50	7137.50	7149.00	7141.3269	60153	8.423772	44
2019–12–27 11:56	7149.00	7142.00	7140.50	7141.50	7140.8169	407108	57.015170	34
2019–12–27 11:57	7141.50	7149.50	7141.00	7141.50	7141.3269	325738	45.615289	22
2019–12–27 11:58	7141.50	7150.00	7141.50	7150.00	7149.4960	390967	54.686307	31
2019–12–27 11:59	7150.00	7150.00	7141.00	7148.50	7142.8571	540701	75.700246	30
2019–12–27 12:00	7148.50	7148.50	7138.24	7138.24	7138.7778	41934166	5874.536073	59
2019–12–27 12:01	7138.24	7138.24	7138.24	7138.24	NaN	0	0.000000	0

Table 2: Head and tail part of the dataset

	open	high	low	close	vwap	foreignNotional	homeNotional	trades
count	282449.00	282449.00	282449.00	282449.00	243964.00	2.82e+05	282451.00	282451.00
mean	9499.88	9502.96	9496.68	9499.87	9558.33	3.69e+04	3.94	23.43
std	1621.16	1622.74	1619.51	1621.17	1632.99	1.37e+05	16.63	46.27
min	6438.00	6443.50	6431.00	6438.00	6432.52	0.00e+00	0.00	0.00
25%	8158.00	8159.50	8157.00	8158.00	8180.62	6.05e+02	0.06	2.00
50%	9542.50	9546.00	9540.00	9542.50	9589.56	7.33e+03	0.77	10.00
75%	10616.50	10619.50	10614.00	10616.50	10656.43	3.20e+04	3.36	26.00
max	14600.00	14600.00	14539.00	14600.00	14547.57	4.19e+07	5874.53	1529.00

Table 3: Basic statistics of the dataset

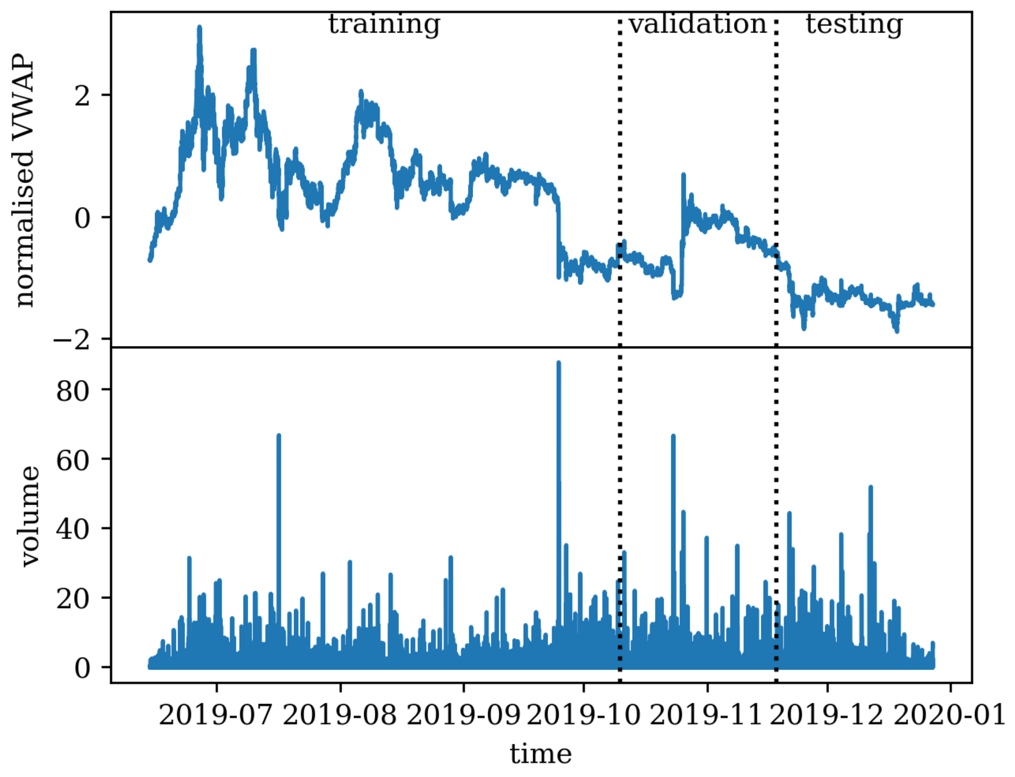

Figure 1: VWAP and volume of the dataset

2.2 Exploratory Visualization

The VWAP and volume of the dataset are shown in Figure 1.

2.3 Algorithms and Techniques

The solution model consists of one LSTM layer and one linear layer. The LSTM layer is formally formulated as follows:

$$\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}\right), \\
f_t &= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}\right), \\
g_t &= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}\right), \\
o_t &= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}\right), \\
c_t &= f_t * c_{(t-1)} + i_t * g_t, \\
h_t &= o_t * \tanh(c_t).
\end{aligned}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $*$ is the Hadamard product.

Moreover, the tunable parameters are listed as follows:

1. `hidden_dim` the number of dimension of the hidden layer between the LSTM and the linear network;
2. `dropout` the dropout regularisation parameter for the LSTM network;
3. `lr` the learning rate;
4. `batch_size` the number of rows for each batch; and
5. `n_epochs` the number of epochs.

2.4 Benchmark

The benchmark model consists of two linear regression layers:

$$\begin{aligned}
h &= \mathrm{ReLU}\left(xA_1 + b_1\right), \\
\hat{y} &= hA_2 + b_2.
\end{aligned}$$

3 Methodology

3.1 Data Preprocessing

Data preprocessing steps are listed as follows.

	from	to
training	2019–06–14	2019–10–09
validation	2019–10–10	2019–11–17
testing	2019–11–18	2019–12–27

Table 4: Dataset splitting

1. **Crawling** Since new data are generating every minute before the futures expires, new rows could be fetched from the data source. The crawler should be able to handle locally cached data and progressively persisting new data.

2. **Column dropping** Columns `open` and `volume` do not provide new information to the rest part and are consequently dropped.

3. **Null filling** For time steps that do not contain any trades, the corresponding `vwap` columns are null. These items will be propagated with last valid observations.

4. **Feature engineering** Moving average convergence/divergence (MACD) with the short period of 12 and the long period of 26, and relative strength index (RSI) with the timeframe of 14, are calculated and appended as extra columns for consequent processing.

5. **Normalisation** All features will be normalised.

6. **Labelling** Each row will be labelled a learning target. The learning target has different definitions in the initial and final solution, which will be illustrated in Subsection 3.3.

7. **Splitting** The entire dataset will be split without shuffling into three consecutive parts for training, validation, and testing, while the lengths proportionate to 6:2:2. Individually, their ranges are listed in Table 4.

3.2 Implementation

The LSTM model was implemented with PyTorch's builtin `torch.nn.LSTM` class. Besides, the training steps are listed as follows.

```
load preprocessed training and validation data into CUDA device;
load parameters;
initialise model;
initialise loss function torch.nn.MSELoss;
initialise optimiser torch.optim.Adam;
initialise scheduler torch.optim.lr_scheduler.StepLR;
foreach epoch do
    split training data into batchs;
    initialise hidden states and cell states with zero vectors;
    foreach training batch do
        clear all gradients;
        execute forward propagation, get estimator, and update
          hidden/cell states;
        get loss value from the estimator;
        execute backpropagation and update the model;
    get loss value of the validation data;
```

3.3 Refinement

In the initial solution, the learning target is the label y, i.e. the VWAP of the next time step, i.e.

$$\hat{y}_t = f(x_t) = \texttt{vwap}_{t+1} + \varepsilon_t,$$

where $f(\cdot)$ is the LSTM model and ε_n is the residual. However, in this solution, a high bias is observed on the testing dataset (as shown in Figure 3). Since the price does not vary significantly in one minute, using the model to estimate the price change in one minute could reduce the bias, which leads to the final solution.

In the final solution, the learning target is defined as the difference of the logarithm of the VWAP between this time step and the next, i.e.

$$\begin{aligned} f(x_t) &= \log(\texttt{vwap}_{t+1}) - \log(\texttt{vwap}_t) + \varepsilon_t, \\ \hat{y}_t &= \texttt{vwap}_t \exp(f(x_t)) = \texttt{vwap}_{t+1} \exp(\varepsilon_t). \end{aligned}$$

Model	Dataset	RMSD
Benchmark	training	209.53
	validation	107.05
	testing	125.28
Solution I	training	147.93
	validation	110.03
	testing	476.31
Solution II	training	11.96
	validation	6.98
	testing	5.53

Table 5: Loss

Even though the estimation $\hat{y}$ is more vulnerable to a higher residual than the former solution, both bias and variance are reduced in the experiment results.

In the following sections, these two solutions described above will be named Solution I and Solution II.

4 Results

4.1 Model Evaluation and Validation

Since the dataset, containing 282452 minutes/rows, is big enough, and the test data was not used to train or tune the model, it could be concluded that the model is robust enough to estimate unseen data.

4.2 Justification

The RMSD losses of three models w.r.t. three datasets are listed in Table 5, which illustrates that the final solution (Solution II) overperforms the benchmark significantly on the entire dataset.

The comparison between ground truth and predictions w.r.t. the linear model as a benchmark and two LSTM based solution models discussed in

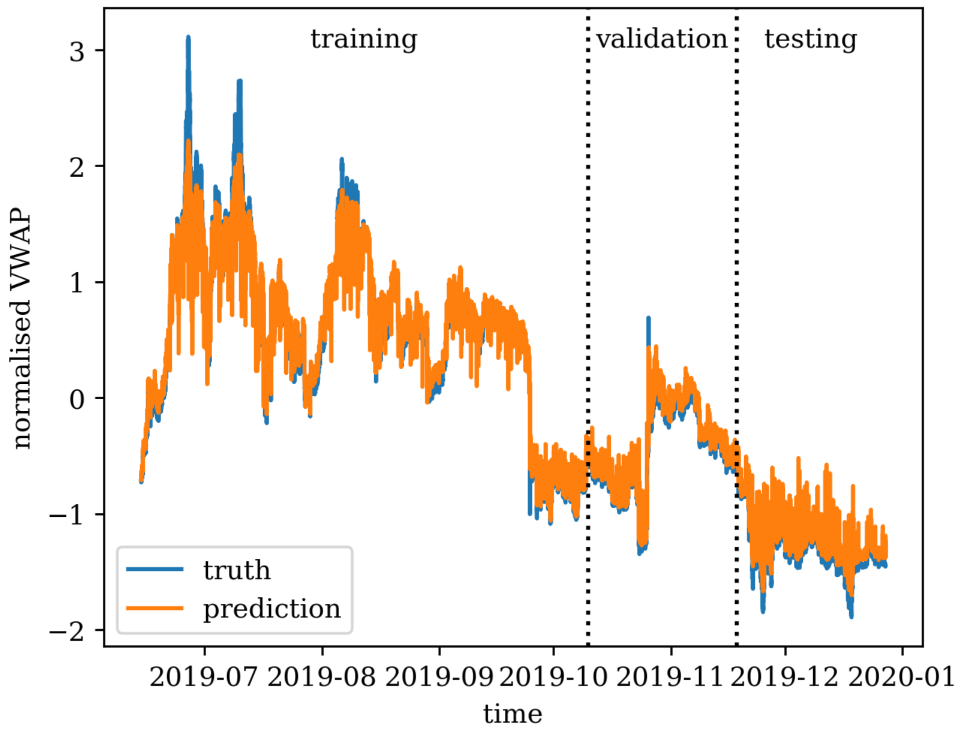

Figure 2: Prediction of Benchmark

Subsection 3.3, are shown in Figure 2, 3, and 4. Based on these three figures we can conclude that

1. The benchmark model performs poorly when price change rapidly, as the orange curve diverges from the blue curve at peaks and troughs;
2. Solution I shows a lower variance than the linear model, but also a high bias in the testing dataset; and
3. Solution II produces an unbiased and efficient estimation on the entire dataset, as the blue curve is almost perfectly covered by orange.

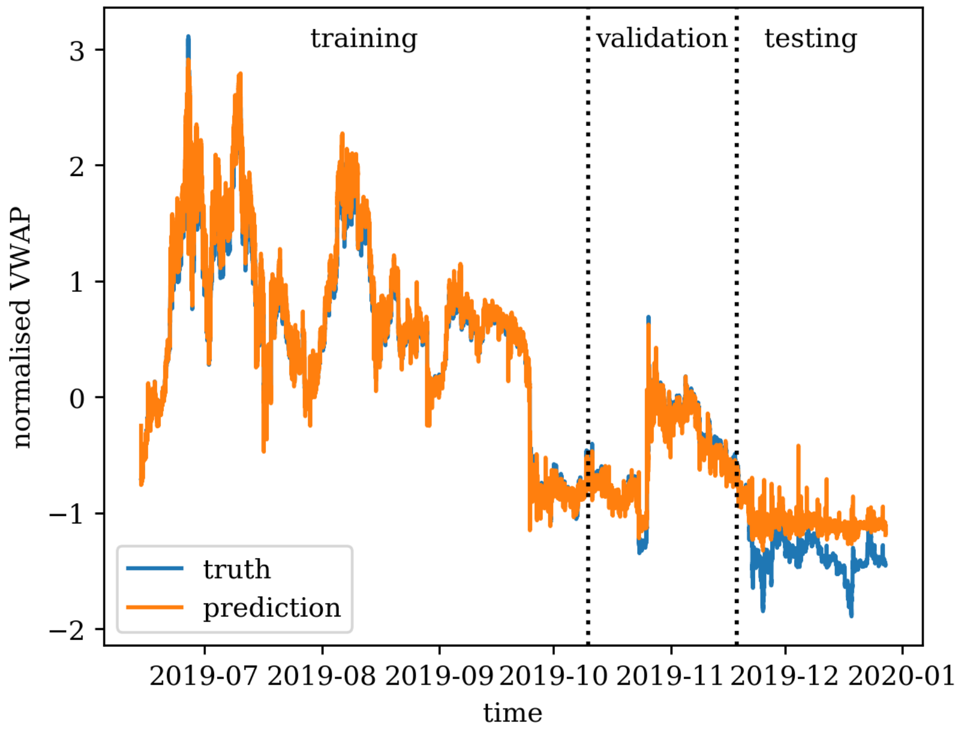

Figure 3: Prediction of Solution I

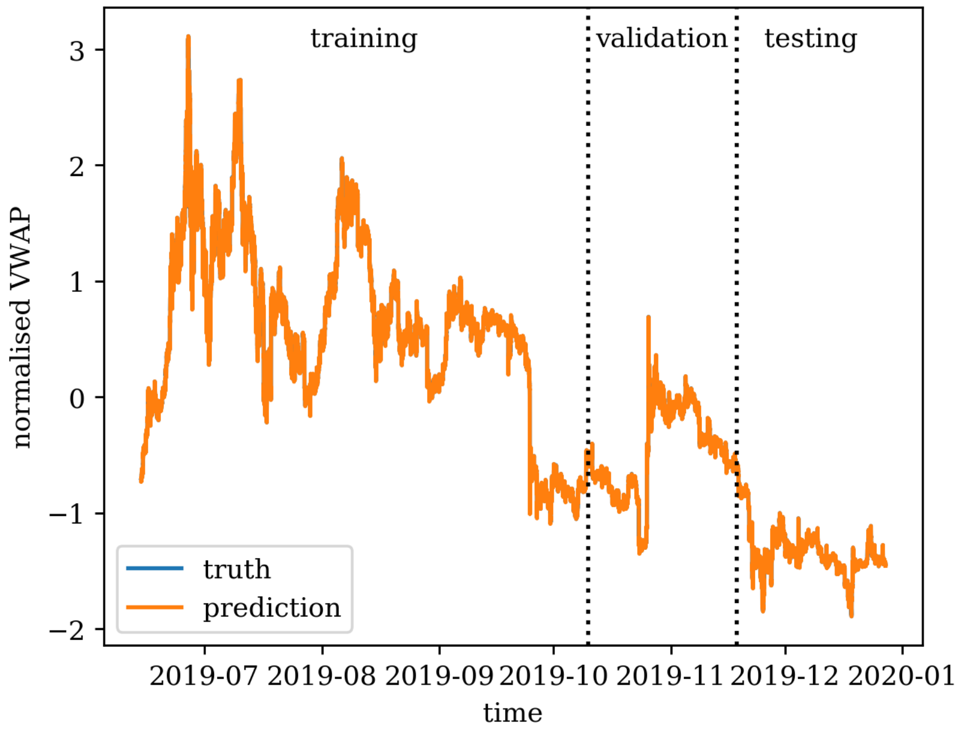

Figure 4: Prediction of Solution II

5 Conclusion

5.1 Reflection

In this study, an LSTM-based predictor was built to estimate VWAP of Bitcoin futures contract using history trading data. The model does not estimate VWAP itself, but the logarithm of VWAP change. Empirically, the RMSD between estimated and real VWAP on the test data is 5.53, which significantly overperforms the benchmark model, whose RMSD is 125.28.

However, even though the estimated price almost overlaps with the real price on Figure 4, further analysis is required to conclude whether this predictor is precise enough to serve as a trading agent that beats the market.

5.2 Improvement

The trading data in financial markets is infamous to have a bad signal-noise-ratio. More data sources outside the market, such as Google Index of the keyword “Bitcoin”, sentiment analysis of the Bitcoin channel on Reddit, could significantly improve the solution of this study.

Moreover, a price predictor could be ensembled with a trading policy and yields a trading agent. The trading policy could be some simple rules or a complex mathematical model. Otherwise, a reinforcement learning model is also possible to serve as a trading agent.