ML Nanodegree Capstone Project
Investment and Trading application

Maciej Nalepa

2020 January

Part I
Definition

Project Overview

Knowing the future of a market stock price is a very valuable information about risk of investment and is commonly pursued because of potentially infinite profits.

Currency ratio and stock markets forecasting based on technical analysis can be easily implemented in a script thanks to a timeseries prediction nature of the problem. Price and volume are publicly accessible and can be used to train a model aiming to output the most likely future price.

In this project I have explored different Machine Learning techniques to find the best predictor of stock market values.

Project was inspired by personal experience in development of trading algorithms.

Problem Statement

We want to predict the price of EURUSD ratio and NASDAQ stock. It is a timeseries forecasting problem. The goal is to create a model that will forecast future values basing on input provided as a starting point. Then a web application will serve forecasts live by downloading information from third party services.

Evaluation Metrics

Because we are developing regression model a norm metric is required. For our purposes we will use Mean Squared Error (MSE) between predicted y and actual values $\hat{y}$ (1). Model loss is calculated from N predictions against actual values.

Evaluation will be performed on test data which is the 20% tail of our datasets. Performance will be compared to a simple MA prediction solution ([1]).

$$MSE = \frac{1}{2}(y - \hat{y})^2 \tag{1}$$

Part II
Analysis

1 Data Exploration

Data is represented with "candles". Each candle is stored in one row of our dataset and it is defined with a datetime stamp. Timeframe represents how long is each candle lifespan, 1 minute being the shortest and technically with no upper limit, but usually it is not common to get data with timeframe longer than 1 month. The dataset I have acquired consists of 5 years NASDAQ stock market history with 1-day timeframe (source: Yahoo! Finance [6]). Another dataset is 5 years of EURUSD history with 1-minute timeframe (source: HistData [7]).

Market data consists of four basic columns: Open, High, Low, Close and for stock markets: Volume.

- Open – the price value when a candle was initiated.
- High – highest price reached during candle lifespan.
- Low – lowest price reached.
- Close – the price at which the candle lifespan passed.
- Volume – the number of shares that changed during candle lifespan.

There are important differences between NASDAQ and EURUSD data. First currency pairs do not have Volume, so this data is missing and for convenience I will drop this column from NASDAQ as well. Another aspect is the value difference, NASDAQ ranges from 4000 to 9000, while EURUSD was much more stable for last 5 years staying around 1.15. EURUSD will be converted to 1-day timeframe, so it matches the other dataset. Such change greatly reduces the amount of entries, which makes working with this data faster (for the cost of information loss).

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-01-02	4760.240234	4777.009766	4698.109863	4726.810059	4726.810059	1435150000
1	2015-01-05	4700.339844	4702.770020	4641.459961	4652.569824	4652.569824	1794470000
2	2015-01-06	4666.850098	4667.330078	4567.589844	4592.740234	4592.740234	2167320000
3	2015-01-07	4626.839844	4652.720215	4613.899902	4650.470215	4650.470215	1957950000
4	2015-01-08	4689.540039	4741.379883	4688.020020	4736.189941	4736.189941	2105450000

Figure 1: Sample of market data (NASDAQ)

2 Visualization

It is vital to have the values correctly distributed. Market is unbalanced and the prices can range from 0 to infinity. For training it is best when the dataset has normal distribution ([3]), the histograms below do show that some kind of normalization will be needed. EURUSD overall looks closer to what we need, but it is possible to get it better than that using normalization techniques.

Market data actually is normally distributed but only on short samples and only if there is no strong trend in the sample (in such case distribution looks more like NASDAQ histograms).

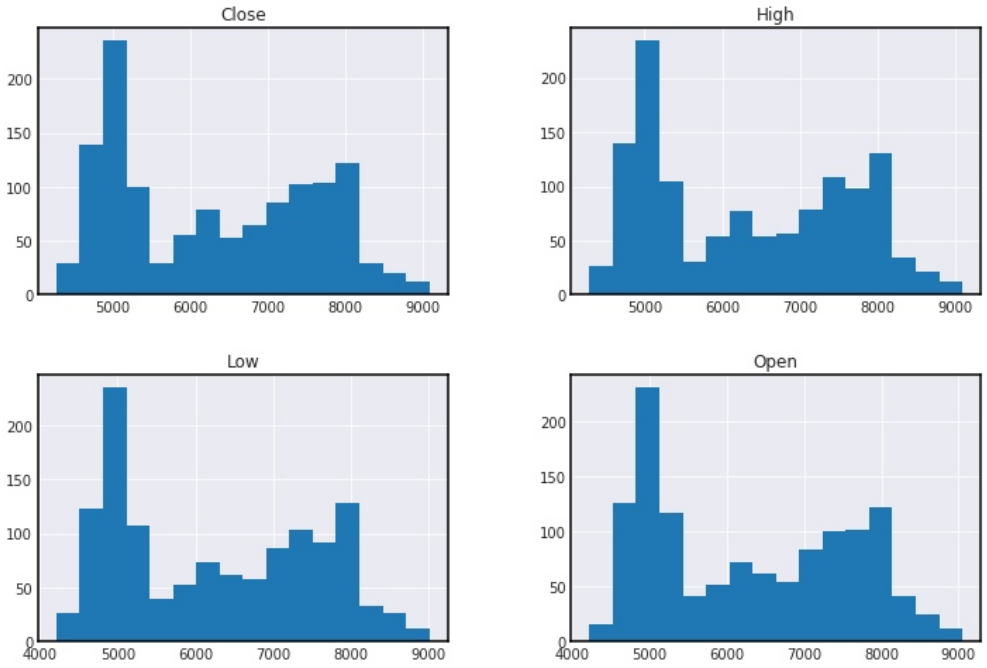

Figure 2: Histograms (NASDAQ)

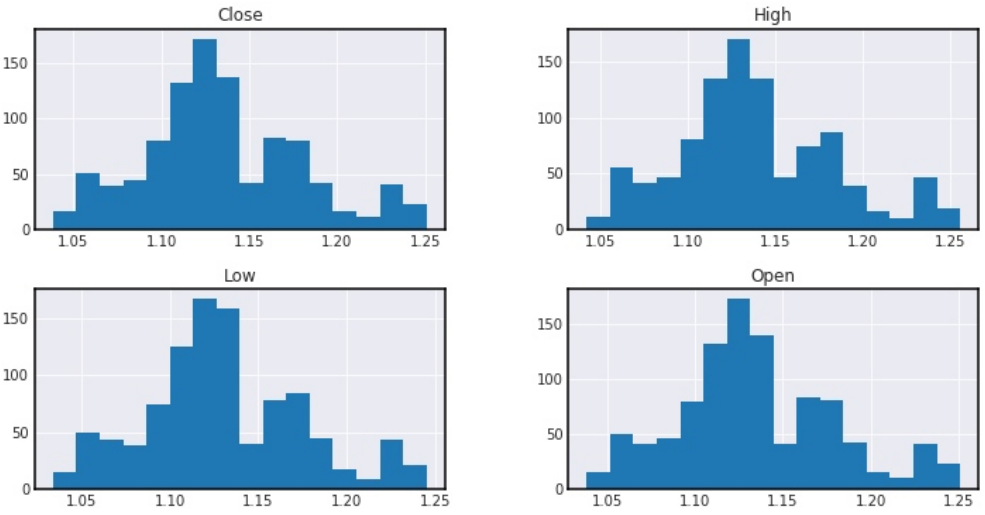

Figure 3: Histograms (EURUSD)

3 Algorithms & Techniques

The regressor uses Long Short Term Memory, which is one of the best currently known methods for working with timeseries data and any other position dependent information sets (e.g. words in a sentence). This model is based on PyTorch framework ([8]).

Architecture of the model is 1-D Batch Normalization over the input split into channels, which are the columns of our data making predictor independent of the input scale. Then 1-D Convolution is applied, which allows to extract more information from the input. Combination of these methods allow the model to work with both datasets in the state as they are – no scaling is required during data preprocessing.

Normalized input is flattened finally passed to fully connected layer, which outputs predictions for N future Close price values. Output is in normalized space though and a function to denormalize output into correct value is required (2).

$$prediction = std(input[Close]) * output + input[Close].last \tag{2}$$

Web application is an HTML document using javascript to retrieve forecasts on demand. User provides any Yahoo symbol as a string and receives forecast for it as a json response. Received data is plotted using Plotly [9].

4 Benchmark Model

The only way to tell how good the model will perform is to compare it with another solution. Moving Average prediction is a great option here. It assumes that next value is the average of last N observations, where N is the count of observed values. It turned out that the value of $N = 2$ is the best. There is a good visualization of how this works [1].

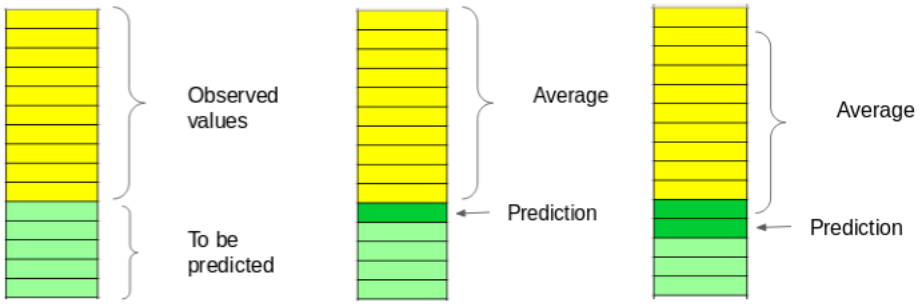

Figure 4: Moving Average prediction sequence [1]

Part III
Methodology

5 Data Preprocessing

As stated in the Algorithms & Techniques section there is no need to scale the data, predictor will do it on its own. Instead of modifying the data it is possible to calculate some indicators knowing only the previous Open, High, Low and Close prices. This way we can generate some new features:

- Gap – days between entries, market is shut down for holidays resulting in gaps in the data.
- EMA – exponential moving average.
- SMA – simple moving average.
- Momentum – value change indicator.
- RSI – relative strength index oscillator.

Figure 5: (NASDAQ) Close price (blue) with EMA (red) and SMA (green) indicators

Figure 6: (NASDAQ) Momentum

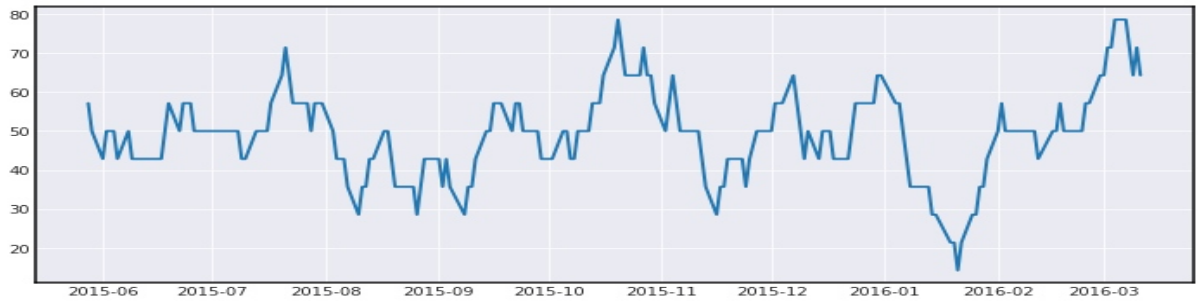

Figure 7: (NASDAQ) Relative Strength Index

6 Implementation

The easiest way to describe implementation steps is to follow the notebooks and specify which scripts they use and what do they do.

In the `Analysis` notebook `utils.py` was imported. It contains definitions for loading the data from `.csv` files and changing data timeframe. To apply some basic indicators a separate module was created for the needs of this project. The module has definitions for indicators SMA, EMA, Momentum and RSI oscillator ([5]). These are enough to get working with the `Analysis` notebook and export datasets into csv train and test files.

The next 2 notebooks are using the `model` directory which is the source dir for both training and deployment. There is `model.py` which contains definitions for LSTMRegressor, SlidingWindowDataset, denormalization function and model_fn. Another files are the required modules and scripts used for training or prediction once deployed.

Next `Model` notebook works with `model.py` and `train.py` files from the `model` directory which is used as a source dir for deployment. This notebook creates benchmark MA model, optimizes it, tests input and training of LSTM model, starts SageMaker training with specified parameters and finally compares predictions losses of the 2 methods directly.

The last notebook is the `WebApp` which basically takes care of getting the correct files into the source dir, tests yfinance module and deploys the RealTimePredictor, so we can use it from the web application. The Webapp is just one HTML file that uses javascript to send requests to the Lambda through API Gateway, there are a lot of steps to get it running on AWS but the HTML itself requires only setting the API url. I wanted to make this more interesting by allowing the webapp to run predictions on any symbol live using a nice Python module for collecting data from Yahoo! [4].

7 Refinement

Currently, the predictor is fixed to use the output size as the horizon of predictions. This means that output size of N will make the model train to predict N future values. This is just implementation because in this project there were no models trained with output size larger than 1.

Part IV
Results

8 Model Evaluation and Validation

After 100 epochs on NASDAQ the loss reached 35382 (train) / 850213 (test), but it was greatly overfitted, because since epoch 30 the test loss was increasing.

EURUSD looked similarly, after 30 epochs started overfitting. Final loss was 0.000365 (train) / 0.001459 (test).

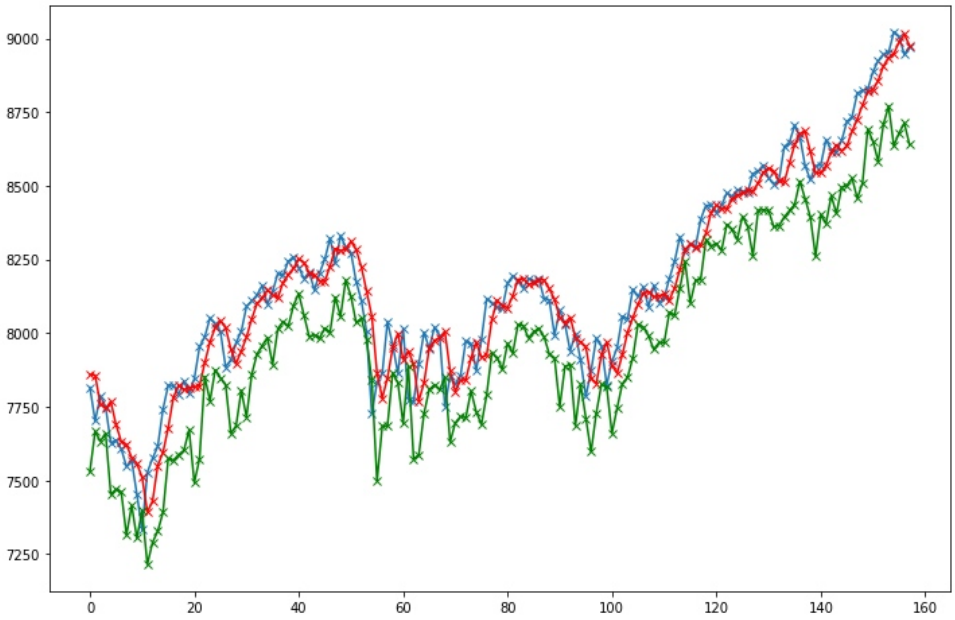

Figure 8: Blue: actual close, red: MA prediction, green: LSTM predicition. Error for NASDAQ was 6.24x the error of benchmark.

9 Justification

Trained model did not perform better than our benchmark. It is caused by the overfitting that was not avoided here (less epochs would do) and also because of the specific architecture that makes it difficult to denormalize outputs.

If some more research was made it can prove possible to create a model that performs better than trivial predictors.

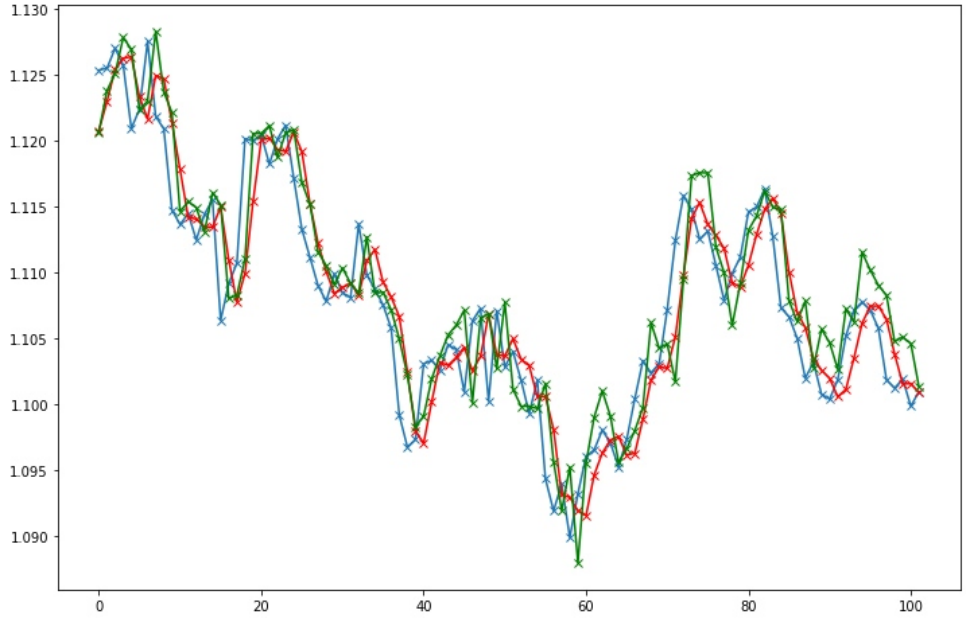

Figure 9: Blue: actual close, red: MA prediction, green: LSTM predicition. Error for EURUSD was around 1.13x the error of benchmark.

Part V
Conclusion

10 Reflection

I guess that efficient-market hypothesis has beaten me once again. That is the biggest challenge of this project, forecasting future market price is not an easy task and can cause headaches.

During development of this project I have run into many problems according the predictor architecture. I could not decide on the best way to deconvolve / denormalize the single value output, solution described above (2) proved to work with best accuracy, but probably it is not the perfect solution. I wanted to avoid scaling the dataset for all costs, because that is a common approach and instead allow the model to do the normalizing over input itself.

11 Further Improvements

Here the same model was trained on both datasets, which did not provide any better accuracy in forecasting. It is easier to train separate model for each symbol.

In this work the datasets had combined length of 2250. More data should improve the model.

Predicting High and Low instead may greatly improve usefulness of the forecasts. It is very difficult to accurately predict a price, but a range can be predicted with a higher certainty.

If the model had accuracy above a certain level it may be profitable in a trading or investing strategy once implemented.

References

[1] Aishwarya Singh, *Stock Prices Prediction Using ML and DL Techniques*, `analyticsvidhya.com`

[2] Yibin Ng, *Machine Learning Techniques applied to Stock Price Prediction*, `towardsdatascience.com`

[3] Rohit Sharma, *Gaussian distribution*, `medium.com`

[4] Ran Aroussi, *yfinance*, `github.com`

[5] Indicators definitions `fxcm.com`

[6] Yahoo! Finance `finance.yahoo.com`

[7] HistData `histdata.com`

[8] PyTorch `pytorch.org`

[9] Plotly `plot.ly`