Report: Predicting Stock Prices with LSTM

Udacity Machine Learning NanoDegree 2020

AJ Miller
March 2020

1. Definition

Project Overview

A stock market is a combination of buyers and sellers who trade a company's shares (STOCKS). Stocks represent a portion of the company, similar to how currency was once tied to a certain amount of gold. Predicting an assets value is difficult because there are many things public or private that may sway the opinion of investors of the value. Such as earnings, lawsuits, overall market stability and the most recent pandemic.

Predicting stock prices or any market's value is a rather popular subject. Individuals and firms have been creating financial models to predict the future for centuries now. The onset of the information age made a plethora of data available, often free, but until recently we couldn't utilize it all let alone efficiently. With the advances in computers and algorithms we are able to use a subset of artificial intelligence (AI), machine learning (ML), to use this information and make more accurate predictions.

We'll be using 10 random and hand picked stocks from the Nasdaq 100 with at least a 10 year history as our training set. This will help reduce the amount of training data, and should reduce noise from IPOs and junk stocks. The list can be found on Wikipedia: https://en.wikipedia.org/wiki/NASDAQ-100

There are many options for getting the stock data, and luckily the data is well labeled. The one I choose is AlphaVantage which offers a free service that includes all data we need and a good history. Some other options are Yahoo! Finance, Bloomberg API, Quandl.

One day of raw data from Microsoft stock (MSFT) https://www.alphavantage.co/

```
"Time Series (Daily)": {
    "2020-03-03": {
        "1. open": "173.8000",
        "2. high": "175.0000",
        "3. low": "162.2600",
        "4. close": "164.5100",
        "5. adjusted close": "164.5100",
        "6. volume": "71577263",
        "7. dividend amount": "0.0000",
        "8. split coefficient": "1.0000"
    }
```

I have been investing in the stock market on my own since after the 2008 crash. Before that all I knew was real estate, and over the years came to the conclusion real estate is for the busy mind but stock investing is for the wise. I also love the idea of AI and what it can, and hopefully will, do for the future of everything. So I choose to combine these two passions for this project.

Problem Statement

The task for this project is to use past data to create a model that can predict the next day's adjusted close. This is a regression problem, which means we are trying to find a continuous value. The problem we are going to solve is looking at some past history points of a given stock and attempting to predict a single future value. The past data (Input) will include opening price (Open), highest and lowest daily price (High, Low), how many stocks were traded (Volume), closing price (Close) and the adjusted closing price (Adjusted Close) which factors in dividends, stock splits and new stock offerings.

Since this is a regression problem that used historical data I suggest comparing a LSTM, which is good at time series forecasting, and comparing that to a traditional Linear Regression Model. The expected outcome is to have the LSTM do better than the benchmark model and have < 5% R-Squared error

Pipeline

- **Load and Explore Stock Data**
- **Split the data into 3 sets training, validation and testing (inseen)**
- **Implement both the benchmark model and LSTM using Keras**
- **Train, check metrics retrain**
- **Compare Results on Both a series and a single time step.**

Metrics

The two metrics used to evaluate the models are R-square and Root Mean Square Error (RMSE). R-squared is a measure in percent of how close the value is to the fitted regression line. RMSE is the average deviation from the true values.

I choose R-squared because it's easy to read 1 to 0 and it gives me an estimate of the strength of the model and the response variable. Since large errors are not desirable I choose RMSE because it shows them better than R-squared.

$$\text{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

R^2: $$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

References: [3]

2. Analyze

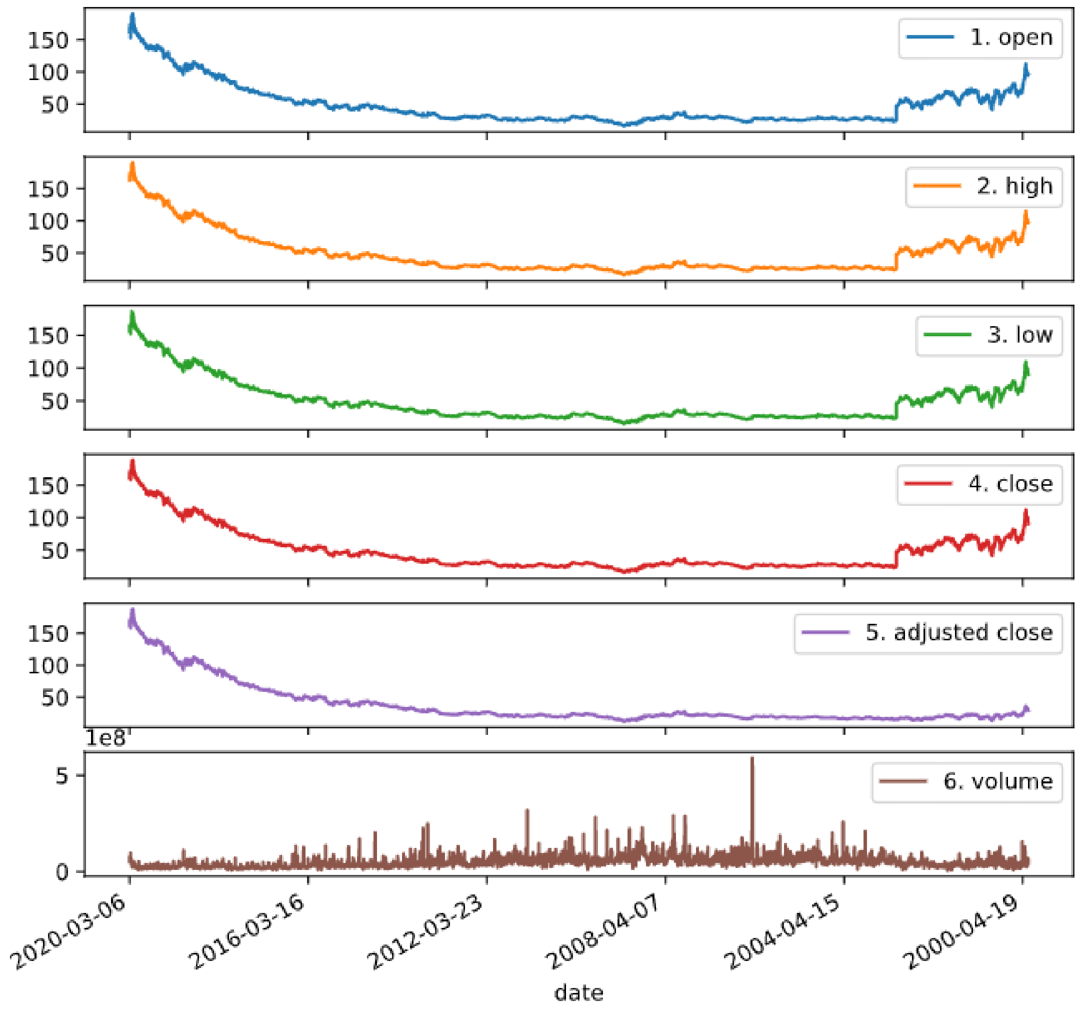
Data Exploration

```
stocks:10
history_size: 20
features: ['1. open', '2. high', '3. low', '4. close', '5. adjusted close', '6.
volume']
time_window: daily
train_feature: 5. adjusted close
Train_feature_index: 4
```

The input data will have 20 days of history each with 6 columns (features). The dividend amount and split coefficient will be removed since that information is included in adjusted close. This means the shape is (20, 6) or as a flat array 120 features. This information will be used to build two models with the same input. It's important to keep in mind how many features you have as that can increase the complexity of your model quickly.

After adding all the stocks and separating train and test we'll have a training set of shape (38836, 20, 6) and a test set of shape (5013, 20, 6). No Abnormalities or characteristics of the data or input that need to be addressed have been identified.

A graph of all the columns and rows of a pandas dataframe for a single stock.



Exploratory Visualization

A single stock example dataset after turning it into pandas dataframe and displaying the first 5 columns. The second one is after we normalize the values and the third displaces mean, std, max

Raw:

```
date          open      high    low       close     adj close    volume
2020-03-06  162.610   163.11  156.00    161.57    161.57        72821057.0
2020-03-05  166.045   170.87  165.69    166.27    166.27        47817251.0
2020-03-04  168.485   170.70  165.62    170.55    170.55        49814383.0
2020-03-03  173.800   175.00  162.26    164.51    164.51        71677019.0
2020-03-02  165.310   172.92  162.31    172.79    172.79        71030810.0
```

Normalized:

```
date          open      high      low       close      adj close    volume
2020-03-06  0.040253  0.039970  0.039038  0.039995   0.046165  0.018048
2020-03-05  0.041103  0.041872  0.041463  0.041158   0.047508  0.011851
2020-03-04  0.041707  0.041830  0.041445  0.042218   0.048731  0.012346
2020-03-03  0.043023  0.042884  0.040604  0.040723   0.047005  0.017764
2020-03-02  0.040921  0.042374  0.040617  0.042772   0.049371  0.017604
```

```
Misc Facts:
      open        high        low         close       adj close   volume
Mean  0.011849    0.011855    0.011849    0.011850    0.010628    0.012203
Std   0.007636    0.007627    0.007636    0.007634    0.009260    0.007056
Min   0.003763    0.003828    0.003721    0.003750    0.003344    0.001450
25%   0.006669    0.006658    0.006694    0.006671    0.005443    0.007361
50%   0.007967    0.007976    0.007995    0.007971    0.006385    0.011006
75%   0.014513    0.014590    0.014469    0.014432    0.011755    0.015203
Max   0.047194    0.046731    0.046663    0.046711    0.053770    0.146486

Total days / rows is 5033.
```

Algorithms and Techniques

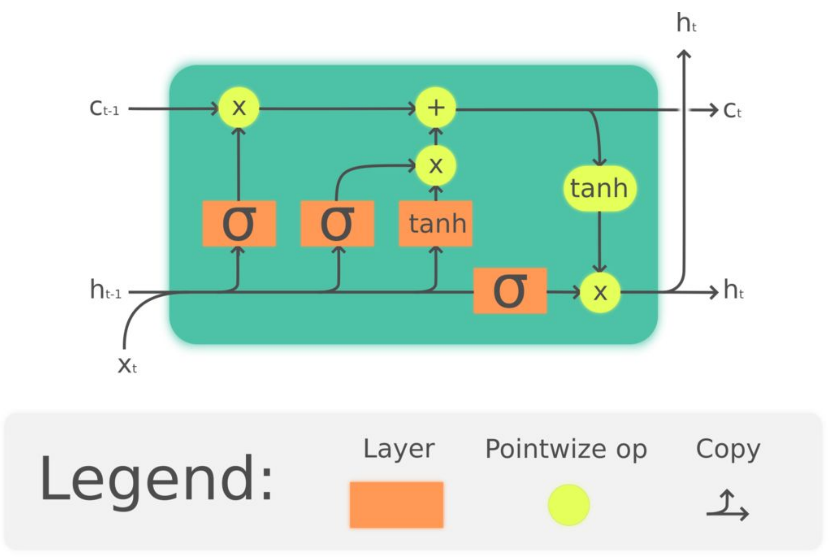
Since this is time series forecasting we'll create and test a Recurrent Neural Network (RNN) with a specialized layer called Long Short Term Memory (LSTM) to predict a stock's adjusted close from the previous month.

The trading days in a month is about 20 so the history size is 20, which is the estimated trading days for 1 month, and a target size of 1. I used the Adam optimizer with a learning rate of 0.001 with a loss function Mean Absolute Error 'MSA'.

Theory

LSTM layers remember the context of the time series data and can act as long-term or short-term memory cells. An example may be if you have a model that learns 1 => 2 => 3 => 4 it may learn that 3's next state is 3 + 1 = 4. So if it sees 1 => 3 => it thinks 4 but should be 5.

They also learn to forget so sometimes past history isn't as important as the last few things it sees. So say after 10 steps in may forget 90% of that information.



By Guillaume Chevalier - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=71836793

Procedure

- **Create a Sequential()**
- **Add the first LSTM layers with 64 units**
- **Add a dropout layer to reduce odds of overfitting**
- **Add the first LSTM layers with 32 units**
- **Add a dropout layer to reduce odds of overfitting**
- **Add Dense layer with 64 units**
- **Final layer is Dense with out 1 which is predicted adjusted close**
- **Compile the model with loss of mean absolute error (MAE) and Adam optimizer**
- **Fit the model on time series data to predict the next adjusted close value**

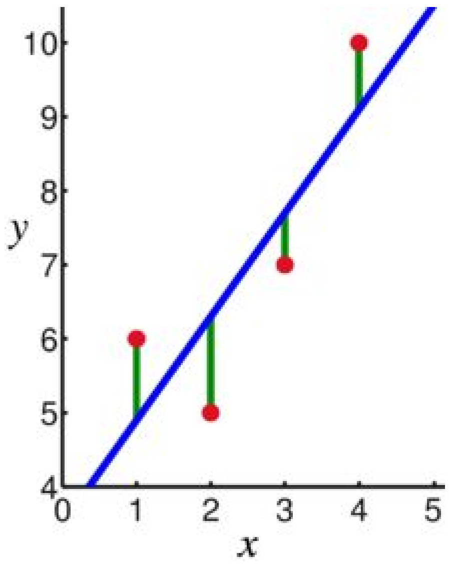
```
Model: "LSTM"

Layer (type)                 Output Shape              Param #
lstm_4 (LSTM)                (None, 20, 64)            18176
dropout_4 (Dropout)          (None, 20, 64)            0
lstm_5 (LSTM)                (None, 32)                12416
dropout_5 (Dropout)          (None, 32)                0
dense_13 (Dense)             (None, 64)                2112
dense_14 (Dense)             (None, 1)                 65

Total params: 32,769
Trainable params: 32,769
Non-trainable params: 0
```

Benchmark

Since this is a linear regression problem, we'll compare our LSTM model to a simple Linear Regression model with 2 dense layers of 100 each and a RMSprop optimizer with a 0.001 with a loss function Mean Absolute Error 'MSA', and the actual market values. We also want to be within 5% of the truth.

Theory

Referring to the image below, linear regression would try to find a relationship (blue) between the (red) observations and the variables y and x assuming the (green) is random deviations.



Procedure

- **Create a Sequential()**
- **Add 2 Dense layers with 100 units each**
- **Compile the model with loss of mae and RMSProp optimizer.**
- **Fit the model on time series data to predict the next adjusted close value**

```
Model: "LR"

Layer (type)                 Output Shape              Param #
flatten_2 (Flatten)          (None, 120)               0
dense_10 (Dense)             (None, 100)               12100
dense_11 (Dense)             (None, 100)               10100
dense_12 (Dense)             (None, 1)                 101

Total params: 22,301
Trainable params: 22,301
Non-trainable params: 0
```

3. Methodology

Data Preprocessing

The first step was to create a list of stocks to gather, so I found a site that listed the Nasdaq that could be copy and passed to google sheets and saved as csv for extraction later. Then I looped through those stocks once loaded into python and used the free Alpha Vantage api and saved them into csv files. The free version of Alpha Vantage only allows 5 calls a minute so I made a function that timed my calls every 20 seconds.

After all data was saved I collected the stocks I would be working with, originally I thought I would need more but 10 seemed fine. Since all stocks trade at different rates and as common practice I normalized the data with keras normalize utility function. This converts all values between 0 and 1 along a given axis.

Implementation

Both models were implemented using Tensorflow 2.0 with keras. I orignial tried to use tensorflows built in dataset but quickly realized I need more experience with pandas and numpy first, their datasets can make a lot of things easier but I was having issues plotting the results. I also had to add a flatten layer to the benchmark model because it was outputting wrong data. I only added dropout layers to LSTM of 0.2 which helps the model from over fitting while training ignores certain 'neurons'.

I also had lots of issues dealing with the different shapes for plotting. I ended up making functions just for plotting the information. Something for me to study is pandas and matplotlib.

References: [2]

Refinement

The first hyper-parameter I tried was the amount of stocks I need, I planned to use all 100 in the nasdaq but started with 20. Shortly after I tried 10 and didn't notice a big change so I kept that. I thought since there are plenty of days in the 20 year history that fewer stocks wouldn't hurt.

The biggest improvement I made was to change both models learning rates from 0.005 to 0.001. I also found the LSTM model didn't need as large layer size as I had originally implemented, and it all helped with training and testing speed. I also added another LSTM layer, which seemed to help. I did add a dense layer to the LSTM after a few runs and seemed to work well.

Originally I started with 2 month or 40 days of history but quickly found that was unnecessary. I reduced it to a month or 20 day history. I also tried many loss functions but the one that always came out on top was Mean Absolute Error.

4. Results

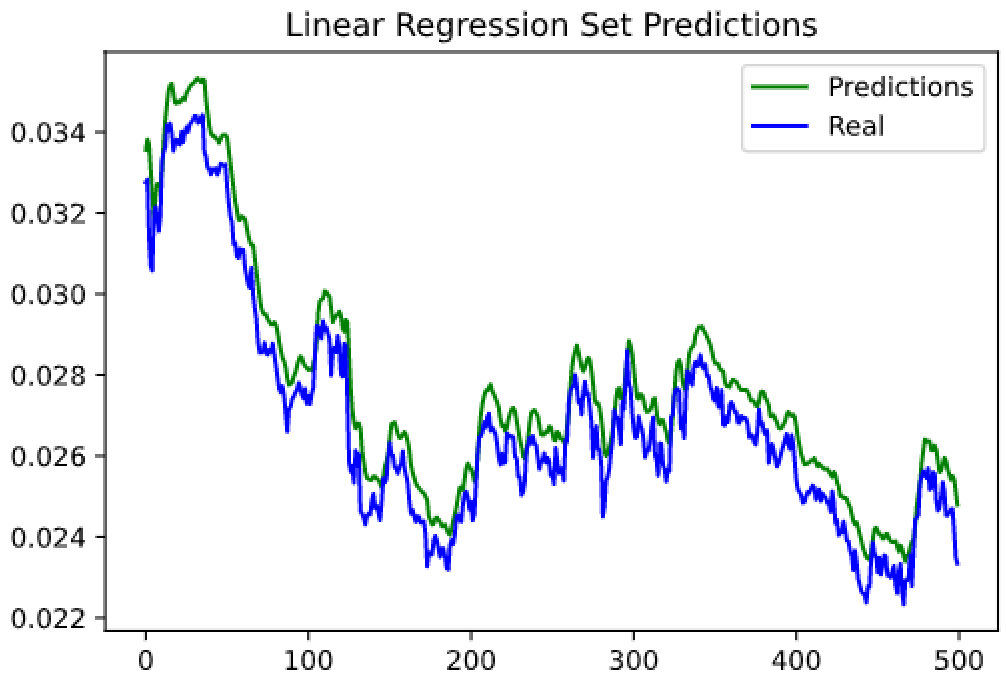
Model Evaluation and Validation

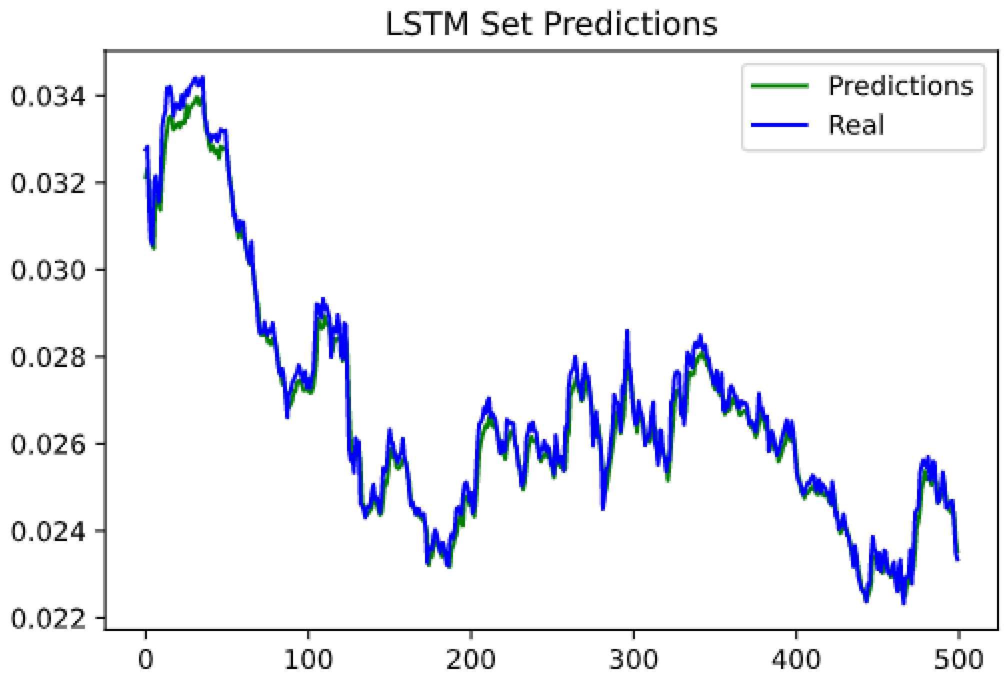
The final model has 3 main layers 1 64 unit LSTM that return its sequence to the second LSTM of 32 units. Then it has a 64 unit dense layer before the final 1 dense output. I kept the training batch size to 32, and the number of epochs to 100.

Justification

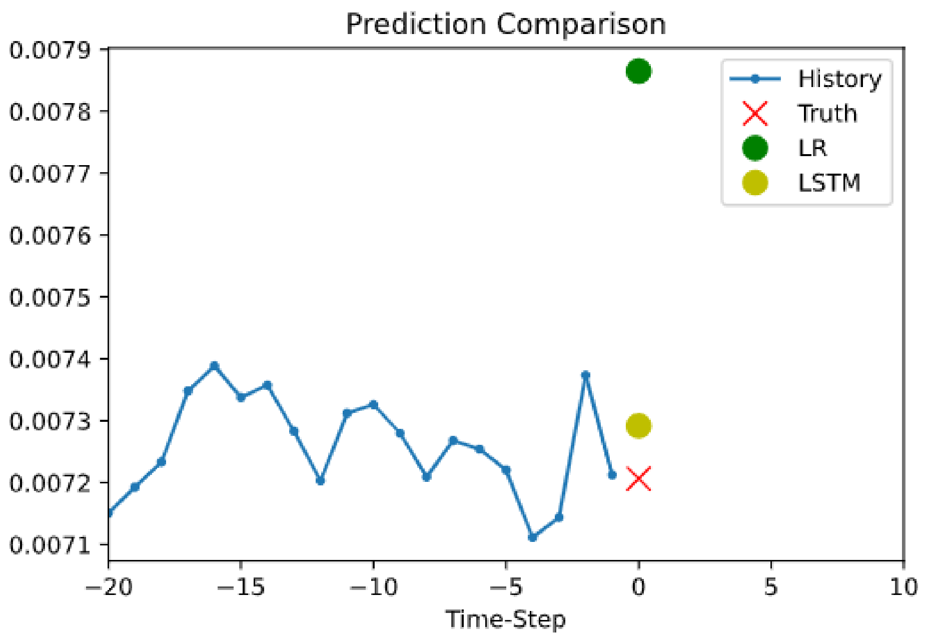
When comparing the Linear Regression Model (LR) and Long short-term memory (LSTM) the scores it clear shows LSTM is doing much better with fewer epochs but took more time. The final score of 98.4% is better than expected, but so is the LR's score of 89%.

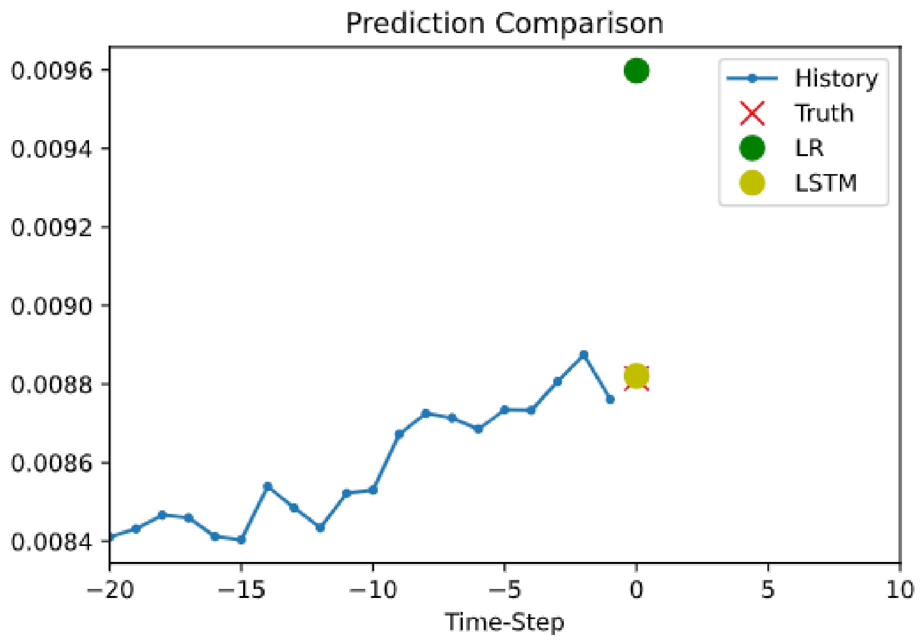
Model	R2	RMSE
Mean	0.0068009325984372815	0.0027989130764170403
LR	0.8954198121511489	0.0009082302620064394
LSTM Test A	0.9716571444618858	0.0004728169761777215
LSTM Test B	0.9555015757615141	0.0005924389119900197
LSTM Final	0.9840881981404725	0.0003542671966026326





The two charts below also show how well the LSTM did compared to the LR. It shows the importance of using different metrics to compare models.

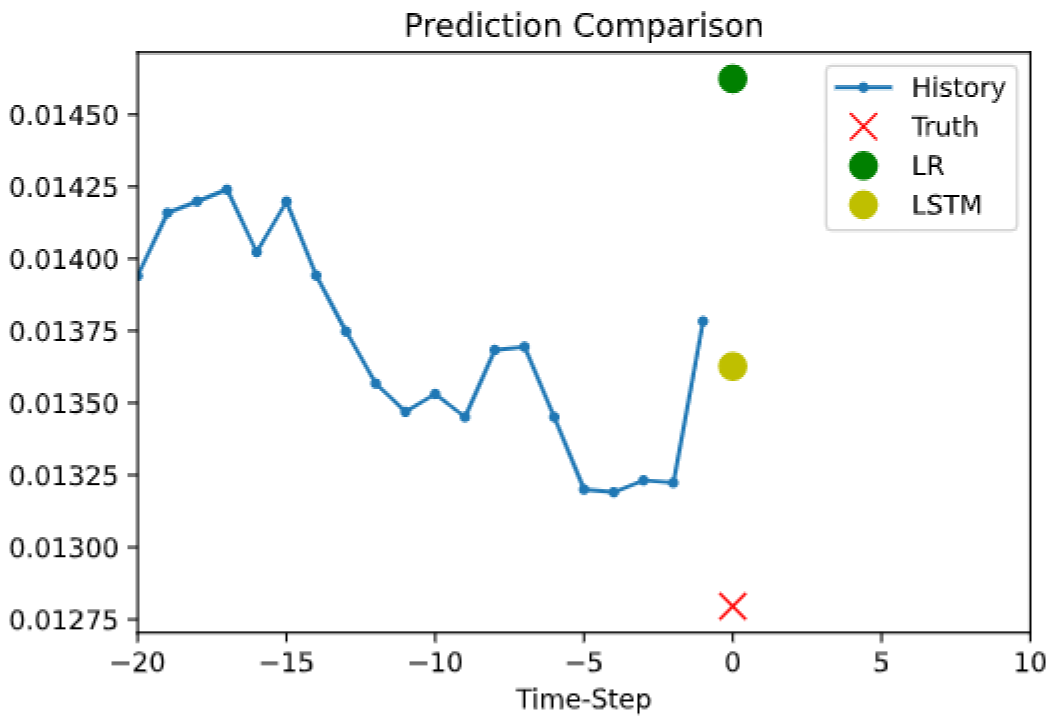




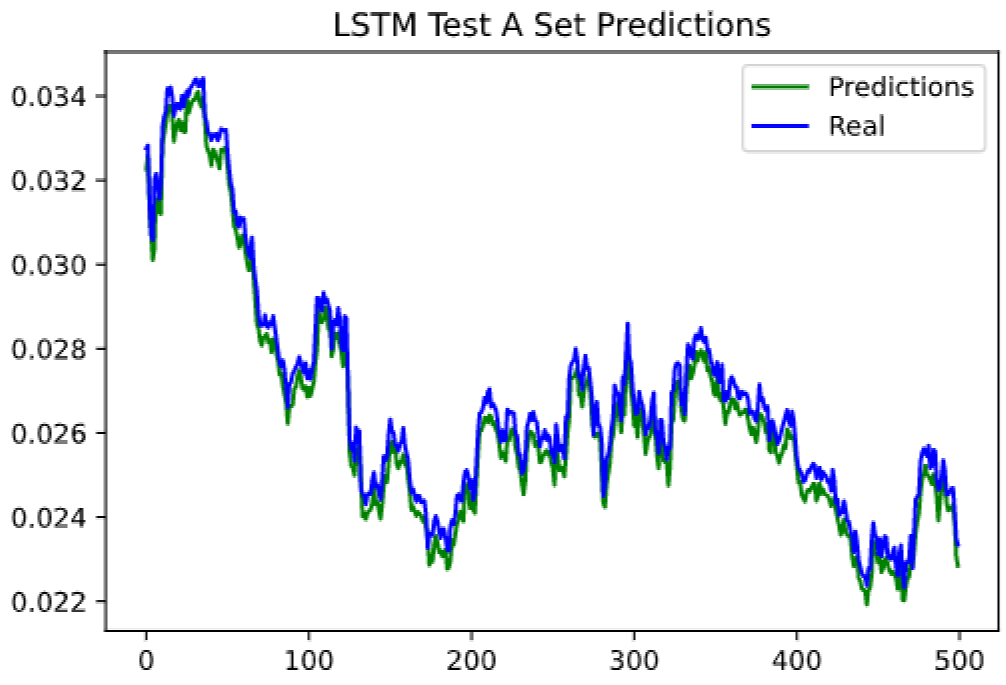
Robustness

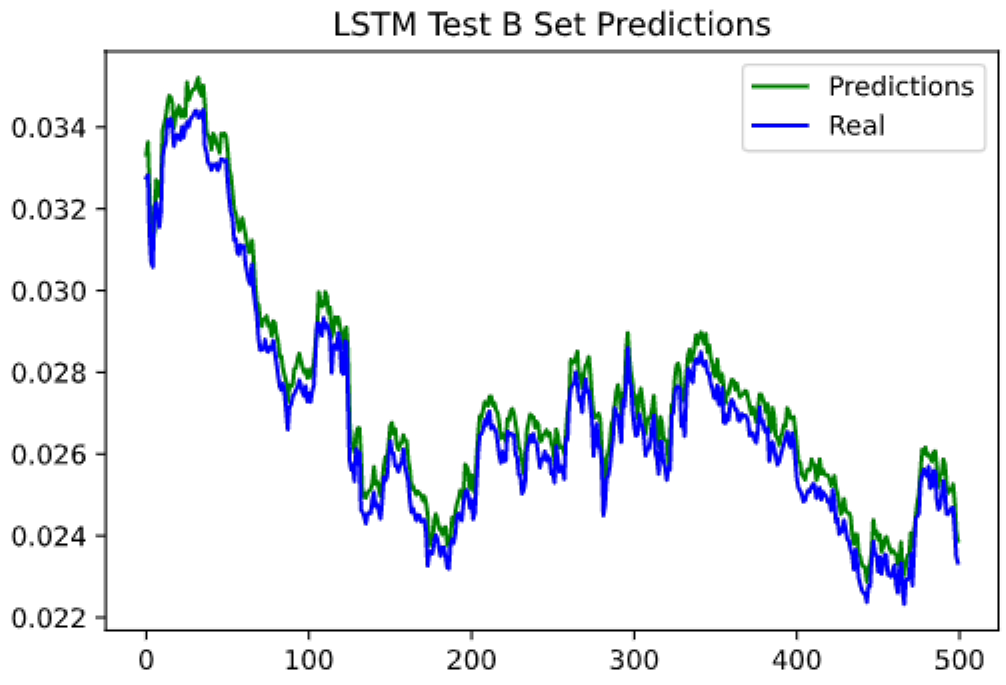
The model seems to generalize well since all charts above were taken from unseen data. It does however seem sensitive to large changes / outliers, such as when a stock moves erratically due to perhaps data that it doesn't know, say earning report from the night before. Overall I feel like the model can be trusted to be within a 5% range if that is acceptable for your task. You may just want to use it as one value in a larger forecasting algorithm.

The other states I tried gave much more error as shown in the table above and the two graphs below. LSTM Test A was given 2 LSTM layers like the final model but no dense layer and trained with 0.005 learning rate. LSTM Test B was given only 1 16 unit LSTM. All other parameters were the same and trained for 100 epochs.



This image shows how when the stock moves unexpectedly , even to the human eye, the model doesn't learn it well.





5. Conclusion

Reflection

The results are better than expected, but training took a long time for the LSTM. Something to consider is whether a more complicated model is always better or could you use a simpler model and achieve the same goal faster.

This whole process was very difficult, and I plan on taking more courses in Machine Learning and perhaps will start with a basic python course. I have learned a lot in a short time.

Improvement

There are many improvements that I could make. I think one of the most important things is to choose your features wisely, so perhaps have some sort of decision tree or model that selects features that matter. I would also like to use multiple models combining results. Perhaps a reinforcement learning model competing with CNN, I've read the CNN's can be used effectively for time series data.

I am certain with enough time, data and knowledge I could make a much better solution. The great thing about Machine Learning is it can always improve based on new data.

6. References

1

- https://en.wikipedia.org/wiki/NASDAQ
- https://en.wikipedia.org/wiki/Stock_market_prediction

2

- https://www.tensorflow.org/tutorials/structured_data/time_series
- https://www.tensorflow.org/tutorials/keras/regression

3

- https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py
- https://en.wikipedia.org/wiki/Coefficient_of_determination
- https://en.wikipedia.org/wiki/Root-mean-square_deviation