Investment and Trading Project

Stock Prediction Project Definition

Project Overview

This is an attempt to predict the stock prices using historical data and applies to the investment and trading domain. While there are multiple factors that can affect stock pricing – economic, political, social etc., we will use only the past stock prices themselves to solve a classification problem (buy or sell) or a regression problem (close price). My main motivation for this project is to learn how to effectively solve a time-series problem with a LSTM as the neural network. There have been multiple works in academia that incorporate Recurrent Neural Networks (RNN)and specifically Long Short Term Memory (LSTM) Neural nets for stock market prediction. I am primarily interested in stock prices of few semi-conductor companies "WIKI/INTC", "WIKI/QCOM", "WIKI/NVDA", "WIKI/TXN", "WIKI/BRCM", and "WIKI/AAPL".

Datasets and Inputs

The input dataset is downloaded using quandl APIs for each of the stocks. This is how the data looks like:
Date,Open,High,Low,Close,Volume,Dividend,Split,Adj_Open,Adj_High,Adj_Low,Adj_Close,Adj_Volume
2017-12-20,174.87,175.42,173.25,174.35,23475649.0,0.0,1.0,174.87,175.42,173.25,174.35,23475649.0
2017-12-19,175.03,175.39,174.09,174.54,27436447.0,0.0,1.0,175.03,175.39,174.09,174.54,27436447.0

Dataset has stock prices since its inception, but I will use only the data starting 2006/1/1 onward.

Apart from the downloaded data from quandl, I will also experiment using some stock indicators such as Relative Strength Index, Average Directional Movement Index, Volatility volume ratio, Simple Moving Average and Stochastic oscillator to create a model that predicts future closing prices.

Problem Statement

All traders do wish to be able to foretell the price of a stock as it amounts to significant wealth (or loss). There have been many attempts to use statistical analysis and machine learning to predict stock prices. However, is it possible? Random walk theory for stock prices states that the price changes of a stock have same distribution and are completely independent of each other and hence the past movement of stock cannot be used to predict the future. However, there is no conclusive evidence that stock prices movement is random and cannot be outperformed. This article [7] argues for both and provides sufficient data to indicate it is not completely random. With that said, we can try to use a Deep neural network to test the theory ourselves.

Here we will attempt to predict future stock prices by using publicly available stock data and using a LSTM neural network. Stock prediction is essentially a time-series sequence prediction problem and the underlying architecture of a LSTM network makes it a good fit.

The main questions to answer are:

1. What is predicted Adjusted Close of a stock based on historical data?
2. What will be the stock price N days into the future?

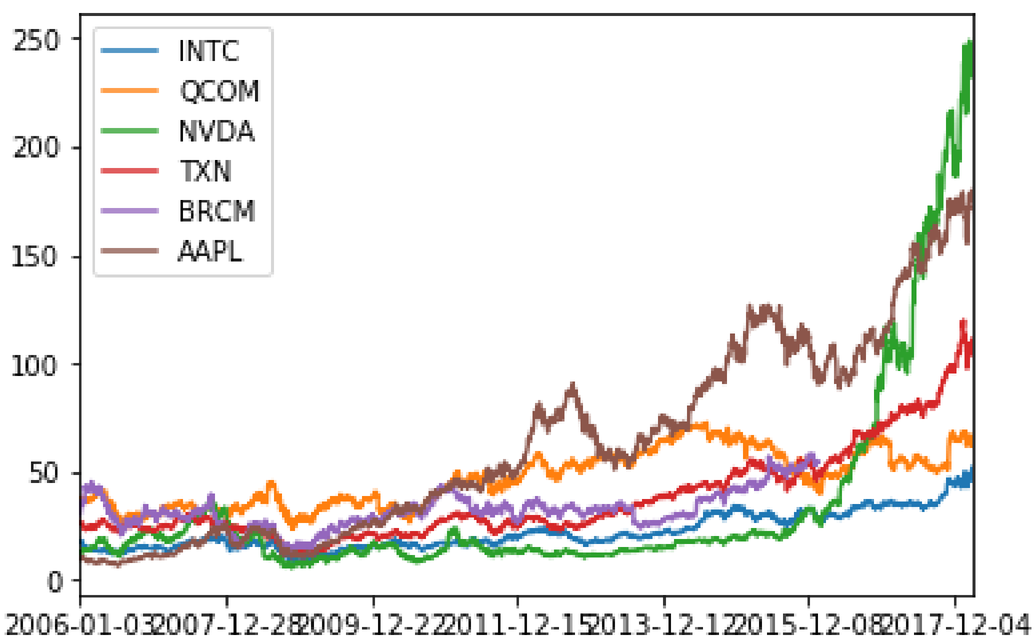
Metrics

The main object of the project is to be able to predict the Adjusted Close price of a stock. This is essentially a regression problem and I will use Root Mean Squared Error metric to measure the performance of the model(s). Here we are evaluating the residual error of the prediction and hence RMSE can give the absolute measure of fit for the model. The unit for RMSE is the same as the input and hence it is more intuitive to understand how well it fits the model. A low RMSE score would indicate a better fit to the model.

Stock Prediction Analysis

Data Exploration

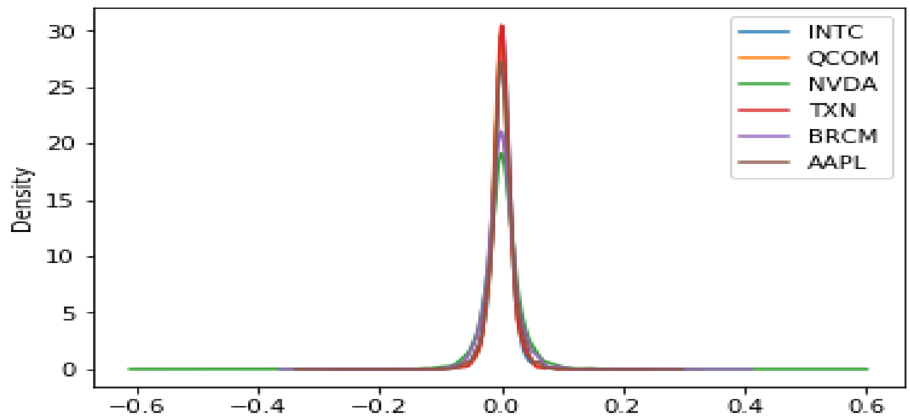
We begin by downloading the stock prices for the interesting stocks and visualize the closing prices/growth of the stock post 2006. 2006 is just a cut-off used since this would be the more relevant data for prediction. Of all the parameters available for a stock, we will try to understand the Adj. Close price since this is the quantity we are going to predict. The data downloaded from *quandl* shows a very strong correlation between the close, open, volume, high, low of a stock on a day to day basis. I also use *stockstats* library to generate a few interesting technical indicators.



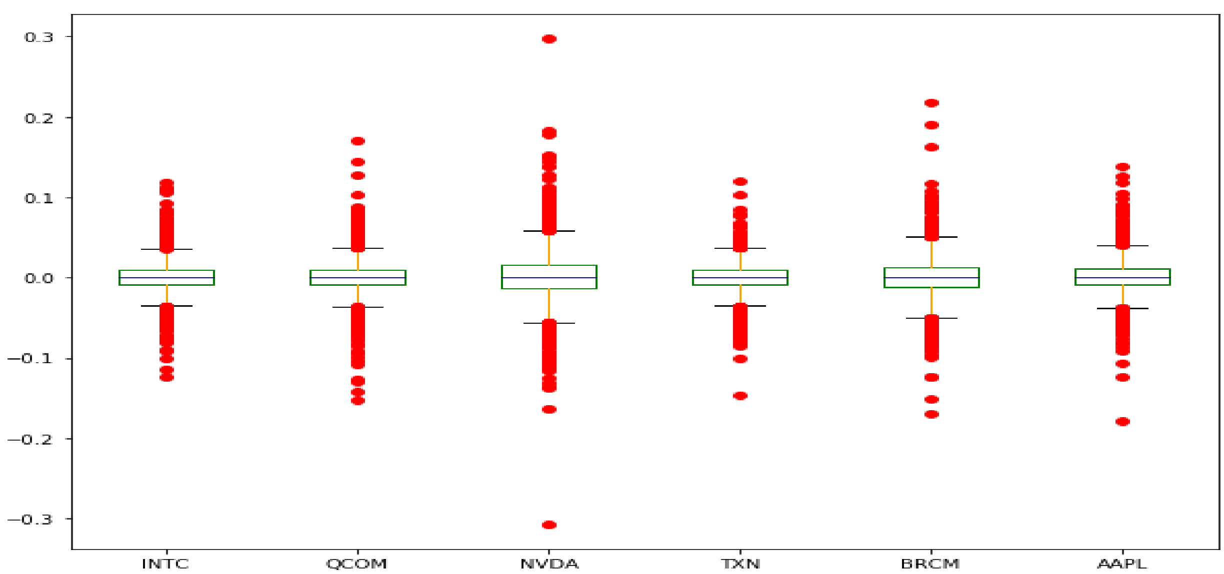
title	INTC	QCOM	NVDA	TXN	BRCM	AAPL
count	3065	3065	3066	3066	2536	3065
mean	21.93807	46.92908	33.57737	37.39588	32.17619	62.72796
std	8.484741	12.78424	46.54202	21.25682	8.747484	45.78066
min	9.052754	23.53545	5.474608	11.25386	12.66826	6.511801
25%	15.54266	34.68717	12.82357	23.69801	26.87465	21.25235

50%	18.732	47.53824	17.36719	28.01993	31.27222	54.28429
75%	29.0066	57.08591	23.91198	47.75263	36.02959	96.69283
max	52.19	72.67418	249.08	120	58.32	179.98

The graph above shows a good description about how each of our stocks behaved. In terms of the closing price of each stock, we see that there are significant differences between the stocks. NVDA clearly shows a significant change in stock prices during the 2006-2018 period (77% of data is the 4th quartile). However, the closing prices themselves do not help much in understanding trends. We should rather use percentage change to understand stock behavior. Plotted below is a box-plot and a density plot to describe day-on-day percentage change for the 6 stocks.



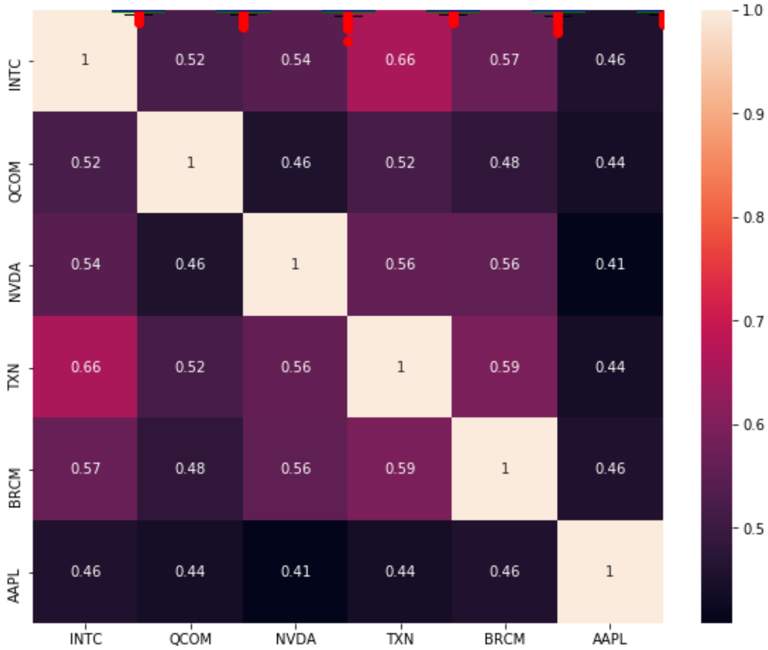
STOCK PERCENTAGE CHANGE DENSITY PLOT



STOCK PERCENTAGE CHANGE BOX PLOT

The general trend of all the stocks seems to be in the 1st or the 4th quartile. It clearly shows that the period between 2006 and 2018 had resulted in significant changes in each of the stocks. There are a few obvious steep changes in closing price for NVDA, BRCM seems to be the only outlier in the data.

The correlation matrix plotted below shows that there is not significant correlation between the different stock prices. (Note: A low value means there is very low correlation). The most seems to be between TXN and INTC but even there it is about 66%. This implies that we should be attempting to model each stock independently. We will create models separately for each stock and hope to predict close prices based on the trained model.



CORRELATION MATRIX BETWEEN STOCKS

More input features:

Investment decisions are primarily based on fundamentals or technical analysis. Fundamental analysis uses financial information about the company whereas technical analysis uses the stock/market information. Since we are using only the stock prices per-se, we will work on a few technical analyses that identify momentum or trends in a stock and see if the combination of a closing-price and these technical indicators help in predicting future stock prices. A few technical analyses used are:

Relative Strength Index (RSI):

RSI is a momentum oscillator i.e. it measures the rate of increase or decrease of stock prices over a period (typically 14 days) and essentially states if a stock is overbought (when above 70) or oversold (when below 30). RSI depends primarily uses the **closing** price of a stock.

Average Directional Movement Index (ADMI):

ADMI is a trend indicator i.e. it basically measures the strength or weakness of a trend and hence provides a better judgement on when to enter and exit the market. ADMI also uses the **closing** price of a stock over a period (14-days typically) using either the observed positive or negative Directional Movement. Unlike RSI, it is a lagging indicator i.e. a trend is set before it can be observed.

Volatility Volume Ratio (VR):

VR is used to measure price range and is an indicator breakout or changes in the price range. It considers the change in **close** prices of a stock along with the **volume** and calculated over a 26-day period.

Simple Moving Average (SMA):

Here we calculate the Simple moving average of **close** price of the stock over 14 days.

KDJ Stochastic Oscillator:

This is a stochastic oscillator that determines the underlying strength and direction by analyzing short term movements. It basically considers **close**, **high**, **low** and **range** of a stock price for a set period (typically 9 days).

The idea is to use the above indicators also as input features to a model and understand if these will help us better predict future stock prices.

Plotting correlation matrix between the technical indicators and the stock closing-prices, we can see that there is very minimum correlation between them ascertaining the fact that we can potentially use the technical indicators also as feature inputs.

Algorithms and Techniques

Stock market prediction is characterized as a time-series problem and hence we need to ensure that model evaluation is done without lookahead bias i.e. the test set a true test set and was never used in training. We will use Sklearn's TimeSeriesSplit to train using different ratios of train and test data. Additionally, we need to ensure that the event sequence is captured and hence we need to avoid randomizing the input samples.

We will measure the performance of the deep neural network against a baseline model to understand if we do need a deep neural network. We will create models to predict one time-step ahead or multi-time step ahead. While using past data to make prediction is necessary, I will also measure how good are the predictions themselves to predict future values. The result would be a model that can predict N days into the future using M historical days (M > N). As part of this project, I will try to predict the stock close prices using (a) a sequence of close-prices (b) a sequence of close-price along with the technical indicators. Note that we will be continuously using only QCOM stock for the rest of our analysis.

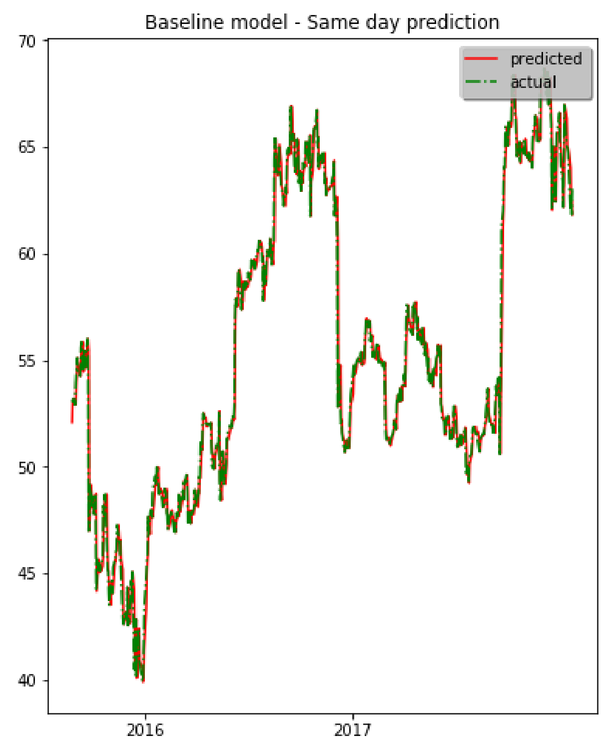
The input feature space has values with different ranges. Hence, we need to normalize the data so that learning will be done in a definite time and that the network will converge to some good minima. We will use MinMax scaler on the train and test sets.

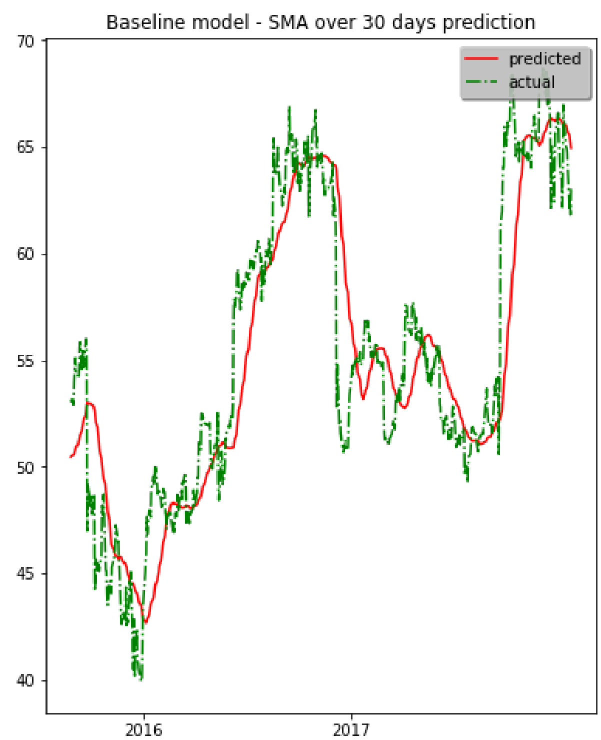
The model will be evaluated using Root Mean Squared Error as this is essentially a regression problem (predict a stock close price in the future). With RMSE, a lower value indicates a better model. Evaluating the performance of a model for a timeseries problem is a bit tricky in that we would like to evaluate it based on its usage. TimeSeriesSplit can be used to generate multiple splits for example, 40/20, 60/20, 80/20 where we ensure test is done on 20% of the sample after the initial train sample. Note that data must not be shuffled so that we continue to maintain sequence. By testing on a constant sample size, we can aggregate the test result to know a more realistic performance of the model.

A more realistic way to evaluate the model is by using a rolling forecast or walk-forward validation. In real world use case, we are more interested in predicting using an online-model i.e. a model which is built on the most recent corpus of data. We decide on a set of samples to be used for training (like the most recent 60 days) and make a prediction for the next (or future) time. With new data, the model is retrained. This is repeated for as many samples required. Note that this would mean we would end up creating many models which is compute intensive.

Benchmark Model

There are multiple ways to design a benchmark model. As part of this project, I would like to opt for two different benchmarks. The first benchmark model will simply predict the current day's closing-price for the next day. While this will perform very well it is not super useful. A 2nd benchmark model will predict closing-price using a simple moving average over past M days. The parameter M needs to be picked based on empirical tests and to some extent on domain knowledge. For comparison against the final model, we do create a TrainTestSplit and evaluate the test data without any need for training.





As can be seen from the graph, the predictions follow the actual closing-price very closely for same day prediction while the SMA prediction follows it with a slight lag. It is clear that increasing SMA over a larger window would show a much larger lag and poorer scores.

Data Preprocessing

Since we are using a LSTM network and a default activation of tanh, I plan to normalize all the input data to a scale of (-1 to 1). Normalization is done so that the deep neural network can learn the weights and biases and will be able to converge to a minima in reasonable time. Furthermore, we need to ensure that the scaling is done separately for the train and test data set to mimic a real-world use case. Since we are learning to predict a sequence using LSTM, there is a need to convert the data to a format that is useable by a LSTM network. The input layer of a LSTM expects data in 3 dimensions i.e. (num_samples, timesteps, num_features) where,

Num_samples - the number of samples/examples we will create

Timesteps - Each timestep includes one observation - so if we wish to use a sequence length of 14 days, this will be the closing-prices for a stock for 14 days in a sequence.

Num_features is the prediction for one time-step. In our case, this will be of length 1 since we are trying to predict 1 future value.

The function *prepareLSTMdata* converts the dataset to feed into the LSTM network. Similarly, when we use the Closing price along with the technical stats about the stock, the multiple variable input data to the LSTM is created using function *prepareLSTMMultiVariateData*

Implementation

As detailed in the data preprocessing step, we first scale the LSTM input data and then create the input to feed to the LSTM network. I use a stacked LSTM model here. The input of the LSTM is a 3D vector of shape (num_samples, timesteps, features) and the output is a 1-dimensional value that is the stock's closing price. Since we are using a stacked LSTM model, the 2nd layer LSTM needs to get a inputs in the form of a sequence. Hence, we set the return_sequences flag in the first LSTM layer for it to return a sequence that can be fed into the next layer.

Model tuning

The first step of the implementation was to identify the best parameters. While there is significant domain knowledge on stock market prediction, I would prefer to choose these based on experiments. For example, how many historical dates to use, how far can we predict to get a decent accuracy. Many other parameters are specific to the model i.e. number of epochs to train on, should we use a stateful LSTM or not, what are the optimizers to use. All of these different combinations have been tried out in the capstone_parameter_tuning.ipynb. The results of the experiments are stored in the consolidated_exp.csv file. The columns of this csv file are as follows:

run_id mHistory-nPredict-Stateful-NumNeurons-NumEpochs-LossFunction-Optimizer-BatchSize where,

mHistory	Number of historical dates to use for evaluation

nPredict	Number of days to predict into the future. Here it is set to 1 i.e. next day prediction
Stateful	Is the model stateless or stateful? Was set to false during experimentation to save on run time as from previous experiments indicated stateful model performed better when using smaller batch sizes.
NumNeurons	Number of neurons to set for the LSTM model. Note that I have already decided it to be a stacked layer since my other experiments showed that a stacked layer did perform better.
NumEpochs	How many epochs to train for to get a performant model
LossFunction	What is the loss function to use? Since it is a regression model, I tried out both Mean Average Error (mae) and Mean Squared Error (mse)
Optimizer	The different optimzer to use.
BatchSize	Size of a batch that is used for training

run1 - LSTM model trained on 1 of 5 folds and tested on 2nd fold

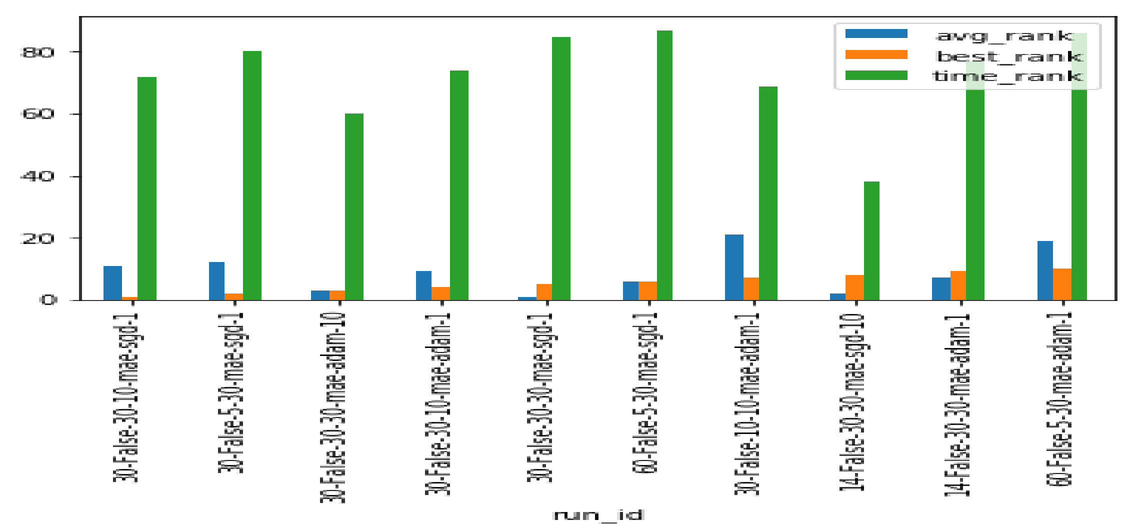
run2 - LSTM model trained on 1st and 2nd of 5 and tested on 3rd fold

run3 - LSTM model trained on 1st to 3rd fold and tested on 4th fold

run4 - LSTM model trained on 1st to 4th of 5 folds and tested on 5th fold

runtime - Time taken to train each model

To pick a model that performed well but also within a reasonable amount of time, I ranked them by best average run (run_avg), best run (run4) and best time (runtime) and plotted a bar chart. **14-False-30-30-mae-sgd-10** seems to have performed well (best run rank of 8, average run rank of 2 and time taken rank of 38. This is the only run in the top-10 whose time taken is within 50 percentile range.



Implementation details:

There are 3 subsections that are covered in this implementation: a) Model Evaluation using next-day prediction b) Single variable (closing-price) prediction for N days into the future c) Multiple variable (closing-price and technical indicators) prediction for N days into the future

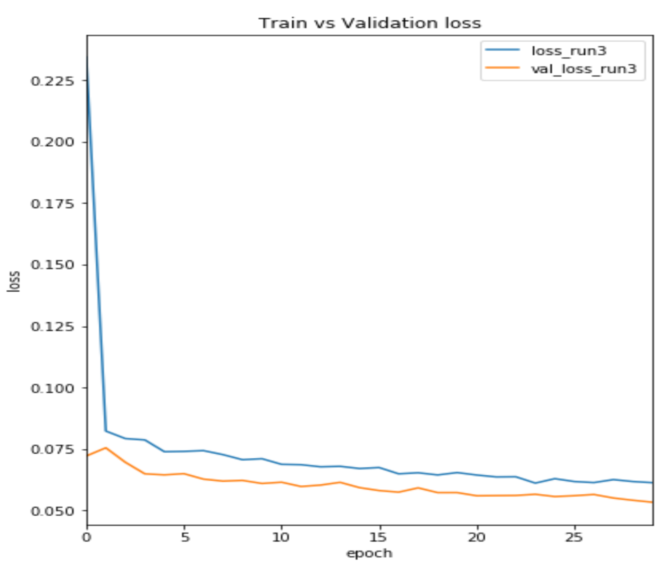
Model Evaluation using next-day prediction

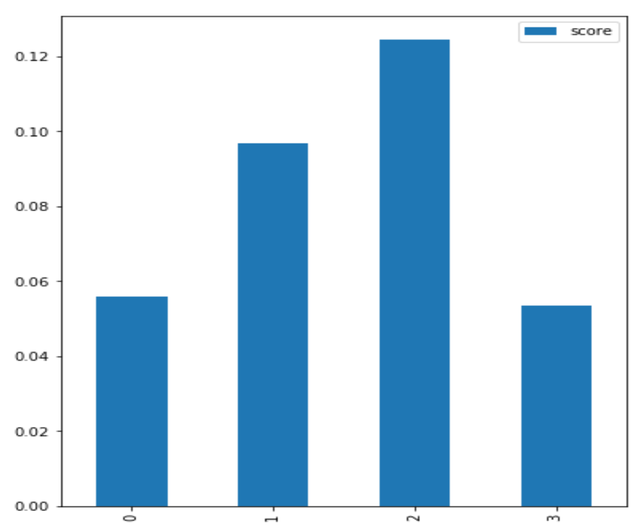
A time-series evaluation cannot be done in the traditional sense of a deep neural network regression model since the sequence or order of events need to be maintained. To be able to evaluate the model, we will use the TimeSeriesSplit and use the train split for training while the test split is used for validation. Using a *timeSeriesSplitCount* of 4, we simulate a 4-fold cross-validation setup as follows:

- Train on 1st fold and validate on 2nd fold
- Train on 1st and 2nd fold and validate on 3rd fold
- Train on 1st to 3rd fold and validate on 4th fold
- Train on 1st to 4th fold and validate on 5th fold

For model evaluation, the output value is the predicted closing-price for the next day. I have experimented using both a stateful and stateless model. While I noticed that the stateful model performed marginally better than the stateless one, the run time was extremely high. While using stateful model, the state of the model will be maintained across batch_sizes. Hence to reset the state after each epoch, we make a call to model.reset_states().

The final evaluation is done on the validation set. While this may seem incorrect, I would like to use a test set that follows the trained sequence and the validation set is the most appropriate. Keeping a held-out data that is too far ahead from the training is also not the most appropriate since we would then be testing on a sequence whose immediate prior data wasn't seen.

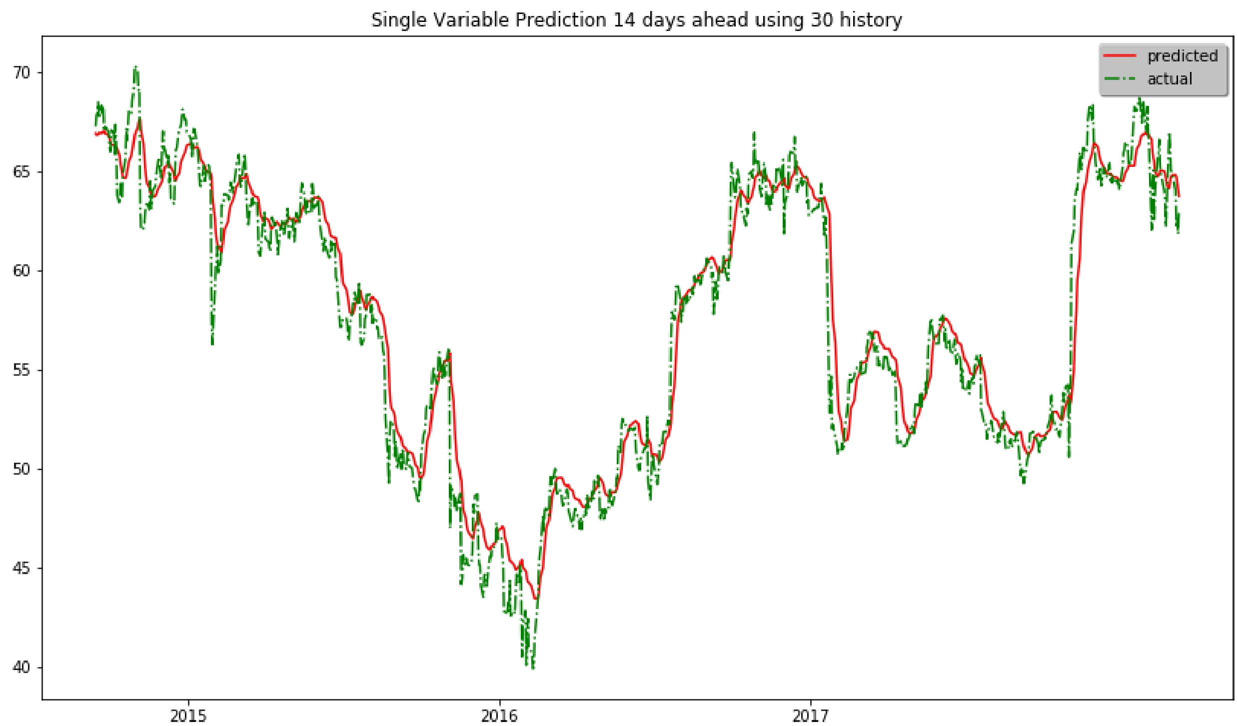




Single variable (closing-price) prediction for N days into the future

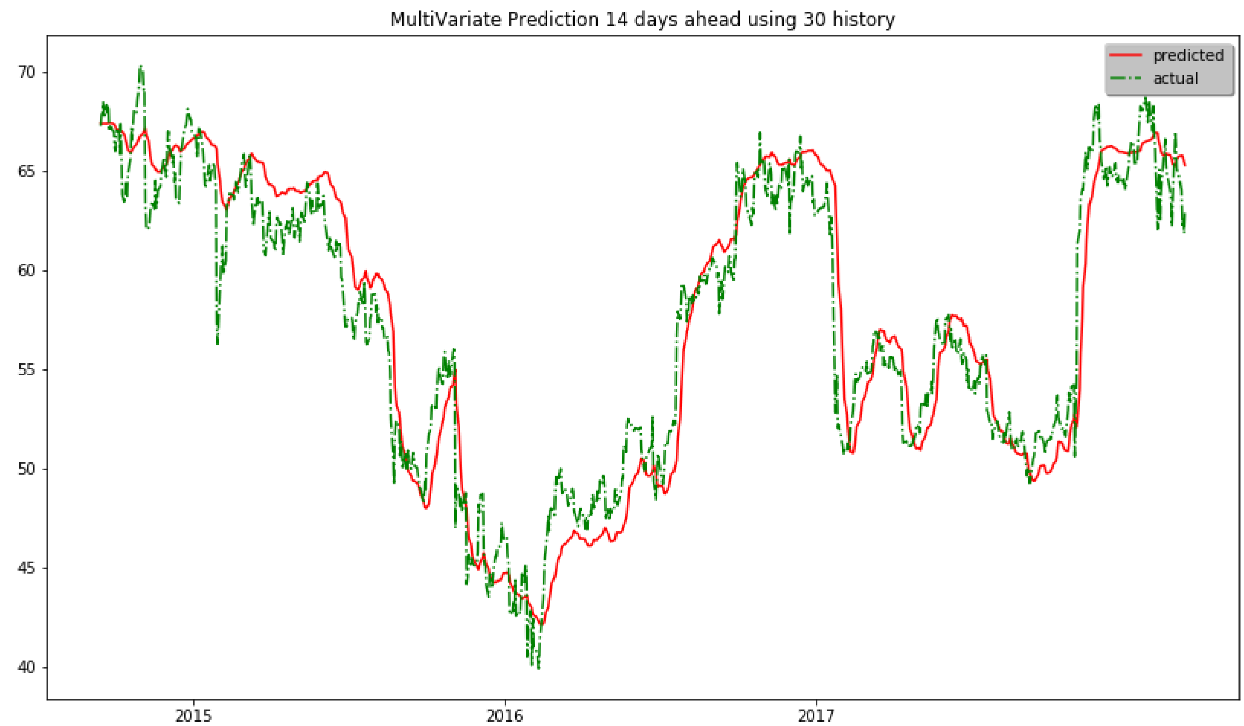
While predicting a day ahead will result in a very good model, it isn't as interesting as if we can predict well into the future. Here we will try to make the prediction for N days into the future (N = 14) using a historical data of M days (M = 30). (N = 14 was our objective and M was arrived at by experiments to find a good model that is developed in reasonable amount of time.)

The base implementation is identical to the code for Model evaluation. However, here we will use a TimeSeriesSplit of 2 to split the data into a train and test set only. The model is trained on the train set. We use a stateful stacked LSTM model as before. The test set is also created in the form of (num_samples, timesteps, num_features). The timesteps is essentially a matrix where the rows are a sequence. For a prediction using 30 days historical data, this is of shape 1x30. However, there is a subtle addition to be done to achieve prediction N days into the future. We use the model to make the next day prediction and then feed this prediction as a next time step in the sequence. In doing so we also drop the earliest timestep i.e. the zeroth one. We repeat this for N times to get the prediction for the Nth day, see function *appendWithPrediction*.



Multiple variable (closing-price and technical indicators) prediction for N days into the future

In the earlier model, we fed a timesteps that contains sequence of only the Closing-price. We can also try running the model using sequences of the closing-price and the four of the five technical indicators. These should be helpful as we noticed that there is very less correlation between the indicators (except for SMA) to the closing-price. Moreover, this information is also available at the current timestep. The major change from the previous implementation is related to the shape of the input to the LSTM model. The num_features in the 3-dimensional array for the Univariate model was of size 1 whereas for the multi-variate it is going to be 5 (closing-price, RSI, StocOsci, ADMI, VVR). The model is still trained to predict the closing-price. Similarly, we need to ensure the same shape is maintained on the test set as well. Below is graph that shows the closing-price for the actual vs prediction N days into the future.



Complications

There have been multiple implementation challenges to getting here. A few listed in the order of complexity:

1. Training time
2. Creating the input data for LSTM model
3. Picking the right domain-based parameters and model hyper parameters
4. Train test and validation for the LSTM model

The inputs for an LSTM model is very different from any other deep neural network models. I had to read up and experiment on the code to begin to understand how to structure the input data. Understanding this helped with the rest of the implementation process. The train/validation/test cycle is also different when working with time-series data sets. Once the basic implementation was done I had to contend with the different parameters that can be used to tune the model. I had to experiment with different parameters to begin to get a grasp of how to feed the data set. One of the more complicated parameters to wrestle with was the stateful parameter. When using stateful=True, we need to be able to set a batch size that is appropriate for the input. This is so because the states are propagated to the next batch i.e. the state of the i'th instance in batch N is propagated to the i'th instance of batch N+1. Using a stateful model showed there was a better result compared to a stateless model though at the cost of much larger training time. The shuffle argument also had an impact on the model performance. While using a shuffle=False [which should be the case for stateful=True] scores better, i notice that it is not the same when using stateful=False. Training on both types of model (stateful set to True or False) was extremely slow on my personal computer but then running it on NVDIA GPUs on the cloud (Amazon EC2 p2.8xlarge) was also time consuming. As part of this project i have already generated the evaluations so that we don't have to regenerate them on all runs.

Model Evaluation and Validation:

While we did evaluate across different train/validation set, I would say the evaluation of the model is best captured by the last train/validation split (since it captures all the sequences known and tests on the most recent known sequence). Model evaluation had been very tricky for this time-series problem since there were two major observations:

When using a stateful model: I observed that there was the graph had extreme fluctuations validation loss between the different runs. If we train for very long, we do see that the training loss seems to be reducing over multiple epochs trained there are some exceptions, which is identical to the validation loss curves. This could be attributed to the fact that for the runs where the validation set is dissimilar to the training set the loss is higher and more representative validation set has lower loss. However, it is difficult to ascertain if the model has high variance (overfit) or high bias (underfit) or underfitting the data from the graphs.

When using a stateless model: The train and validation loss graphed suggested that model had been improving over multiple iterations. Both train and validation loss reduced drastically till about epoch=3 and then continued to reduce at a much slower rate beyond epoch=5. It is obvious the model does not underfit the data. The model does not overfits the data since the validation loss continues to reduce along with the training loss. [For an overfit model, we should be seeing the validation loss increase after the meeting point of the train and validation curves]. It is likely that running it for more iterations could give a better judgement call. Also, having used shuffle=False may have allowed the model to generalize well for unseen cases. However, to conclude that it is a good fit may not be entirely true since we don't see the curves getting any closer. I think creating multiple feature inputs that can better model stock price would be a good addition to create a LSTM model that generalizes well for the input.

Justification

While the model evaluation and validation seemed to be a damper, we can see that the final two models that were used to predict for N days into the future showed that we can perform much better than the baseline model.

Baseline - Predict using current day closing-price price: 0.88

Baseline - Predict using SMA of closing-price: 2.50

Model Evaluation - Predict using LSTM for next day based on 30-day: 0.05

Historical days used	Predict N days head	Univariate LSTM	Multivariate LSTM
14	1	2.02	2.03
30	14	1.72	2.24
60	14	1.72	2.32

It is evident that we cannot reach the best baseline score - since this is predicting only a single day in advance. Our evaluated model which was configured to predict next day using 30-day history does show better scores (using the last fold of the validation set) indicating that the LSTM can perform as well as a baseline algorithm. I think the main reason why the evaluated model performs well is that unlike the best baseline model, it can use the sequence but also weigh heavily on the more recent values of the sequence.

However, a realistic usage begs us not to use a next day prediction and hence the two other LSTM models that were trained based on different historical dates and parameters. We see that the univariable LSTM model [model that used only sequences of closing-price] performs better than the SMA baseline algorithm even though it cannot match the next-day prediction baseline algorithm. I believe this is because it is behaving more like the SMA baseline algorithm (the prediction line on the graph lags the actual value) but with better understanding of the sequence. It is a similar case with the multi variable LSTM model [model that uses both closing-prices and technical indicators in the input sequence]. The lag is more prominent in the multivariate LSTM model when the historical dates to use is increased. The score changes over usage of historical data and how far ahead to predict does make an important case i.e. stock market can be modeled and given more intelligent use of the data we should be able to get more accuracy.

Improvements

We saw that technical analysis can be used as input. Similarly, we could use a stocks fundamental. This would be an interesting case-study as it would consider the financial health of the company which is a strong indicator of how well it can perform in the future. Unfortunately, Stock markets are also driven by the human ideas and motivations - hence we would need to find a suitable way to provide this as inputs. Twitter feeds and news reports are a good source to model human motivations. For an end-user, sometimes, the actual close-price may not be very useful compared to predicting stock decision labels such as buy / sell / hold. We can feed the output of the regression model into a classifier that can then predict these labels. On a more interesting note, we could create a hybrid model that feeds on the current developed model and a 2nd model that is modelled based on tweets, news, fundamentals and use the hybrid model to predict the label.

Conclusion

We have successfully created a deep neural network-based model that can predict closing-price of a stock. To achieve this, a baseline algorithm was initially implemented to compare against the final model. Different techniques were used to identify good domain parameters and model hyper-parameters to achieve our objective. Graphical visualization tools were used to understand the input features and the final output. One of the more interesting things I learnt in this project was that the approach to a sequence prediction problem seems very different from the other regression or CNN problems. From data preprocessing to model building to validation and testing, every step was completely different from what I had learnt during the course. One another aspect of the project which I found interesting was the use of technical indicators as the input. I wasn't sure how this would turn out especially considering that most technical indicators were derived from the closing-price of the stock itself. I was pleasantly surprised that a multi-variable LSTM does score as well or better than a single variable LSTM model. I am extremely thankful to the numerous research articles and blogs that helped me get to this point with my project.

References:

[1] http://snap.stanford.edu/class/cs224w-2015/projects_2015/Predicting_Stock_Movements_Using_Market_Correlation_Networks.pdf

[2] http://ieeexplore.ieee.org/document/7364089/

[3]
https://etd.auburn.edu/bitstream/handle/10415/5652/Application%20of%20machine%20learning%20techniques%20for%20stock%20market%20prediction.pdf?sequence=2

[4] https://arxiv.org/pdf/1801.04503.pdf

[5] LSTM for house prices: https://arxiv.org/pdf/1709.08432v1.pdf

[6] Stateful LSTM: philipperemy.github.io/keras-stateful-lstm/

[7] Share prices and Random Walk :
https://www.tcd.ie/Economics/assets/pdf/SER/2007/Samuel_Dupernex.pdf

[8] https://www.investopedia.com/university/technical/

[9] http://www.binaryoptionsthatsuck.com/kdj-stochastic-indicator-can-you-improve-on-perfection/

[10] Rolling forecast: https://www.otexts.org/fpp/2/5/

[11] TimeSeries Validation: https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

[12] http://francescopochetti.com/pythonic-cross-validation-time-series-pandas-scikit-learn/