Machine Learning Engineer Nanodegree
Capstone Project

Stock Price Prediction using Machine Learning

Samarth Mothakapally

December 24, 2020

Definition

Project Overview

Predicting stock prices has been a widely popular domain since the establishment of the trading exchanges in 1792. Accurate predictions help in minimizing the risks. Since the last few decades, conventional statistical methods, like time-series and multivariate analysis, were being used for making such predictions. In the last few years, with the advent of cloud computing and several improvements in running the machine learning algorithms efficiently at scale, machine learning methodologies, including artificial neural networks, genetic algorithms (GA), and fuzzy technologies have been used to predict stock prices.

In the paper "Support Vector Machines for Prediction of Futures Prices in Indian Stock Market "[1], authors discuss Back Propagation Neural Network and Support Vector Machine techniques to construct the prediction models for forecasting the five significant futures indices of Indian markets. They found the normalized mean squared error (NMSE) to be in the range of 0.929 to 1.152 while using SVM, which is pretty decent.
In the paper "A deep learning framework for financial time series using stacked autoencoders and long-short term memory "[2], authors present a novel deep learning framework where wavelet transforms (WT), stacked autoencoders (SAEs), and long-short term memory (LSTM) are combined for stock price forecasting. The performance is reported across a year's worth of data across various types of markets and reports Mean absolute percentage error (MAPE), correlation coefficient (R), and Theil's inequality coefficient (Theil U). They predict the S&P 500 index in the developed market, with an average value of MAPE and Theil U of WSAEs-LSTM as 0.011 and 0.007. It looks like ML and Deep learning methods yield good results in stock prediction.
I have always been intrigued by the stock market and have been mainly concentrating on fundamental analysis. Leveraging ML to build a model to predict stock prices will enable me to understand stock movements better if the technical analysis substantially correlates with a company's fundamentals.

Problem Statement

For this project, the goal is to build a Machine learning model that will predict the stock price using historical time series data. The model will take the daily trading data over a specific date range as input includes the Open, Low, High, Close, and Volume data and predicts adjusted close for the given query dates.

Metrics

As this is a regression problem, the metrics used would be R-squared and root mean squared error (RMSE).
R-squared is a statistical measure of how close the data are to the fitted regression line.

R-squared = Explained variation / Total variation
In general, the higher the R-squared, the better the model fits your data.
Root-mean-squared-error is the average deviation of the prediction from the actual value, and it can be compared with the mean of the actual value to see whether the deviation is large or small. We can then use the RMSE to measure the y values' spread about the predicted y value. In general, the lower the RMSE, the better the model fits your data.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values
$y_1, y_2, \ldots, y_n$ are observed values
n is the number of observations

Analysis

Data Exploration

S&P 500 consists of 505 companies. The list of the symbols is saved into the file s&p_tickers_list.csv. Historical daily trading data is downloaded for these symbols using the yfinance package.
The data range is from January 1, 2010, to November 30, 2020, and the data will include all stocks' Open, High, Low, Close, Adj Close, and Trading Volume. A sample of a stock's data is below.

Apple sample historical trading data

```
data_raw[:6]
```

Date	Open	High	Low	Close	Adj Close	Volume
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.593426	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656428	6.604825	601904800
2010-01-06	7.656428	7.686786	7.526786	7.534643	6.499768	552160000
2010-01-07	7.562500	7.571429	7.466072	7.520714	6.487752	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.530883	447610800
2010-01-11	7.600000	7.607143	7.444643	7.503929	6.473271	462229600

After downloading the trading data for all the stocks and flattening the dataframes, sample data is below.

Date	Adj Close_AAPL	Adj Close_MSFT	Close_AAPL	Close_MSFT	High_AAPL	High_MSFT	Low_AAPL	Low_MSFT	Open_AAPL	Open_MSFT	Volume_AAPL	Volume_MSFT
2010-01-04	6.593426	24.105360	7.643214	30.950001	7.660714	31.100000	7.585000	30.590000	7.622500	30.620001	493729600.0	38409100.0
2010-01-05	6.604825	24.113148	7.656428	30.959999	7.699643	31.100000	7.616071	30.639999	7.664286	30.850000	601904800.0	49749600.0
2010-01-06	6.499768	23.965164	7.534643	30.770000	7.686786	31.080000	7.526786	30.520000	7.656428	30.879999	552160000.0	58182400.0
2010-01-07	6.487752	23.715933	7.520714	30.450001	7.571429	30.700001	7.466072	30.190001	7.562500	30.629999	477131200.0	50559700.0
2010-01-08	6.530883	23.879499	7.570714	30.660000	7.571429	30.879999	7.466429	30.240000	7.510714	30.280001	447610800.0	51197400.0
2010-01-11	6.473271	23.575750	7.503929	30.270000	7.607143	30.760000	7.444643	30.120001	7.600000	30.709999	462229600.0	68754700.0

The data is stored in trading_daily_s&p500.csv. There are 3036 columns, and 342 columns have missing values. It could be due to ticker added to S&P500 after 2010, removed from S&P500, mergers, etc. Because the neural network cannot accept missing values as input, it's impossible to impute these missing values because they do not exist. I will be dropping these columns. Hence the total number of columns would be 2694.

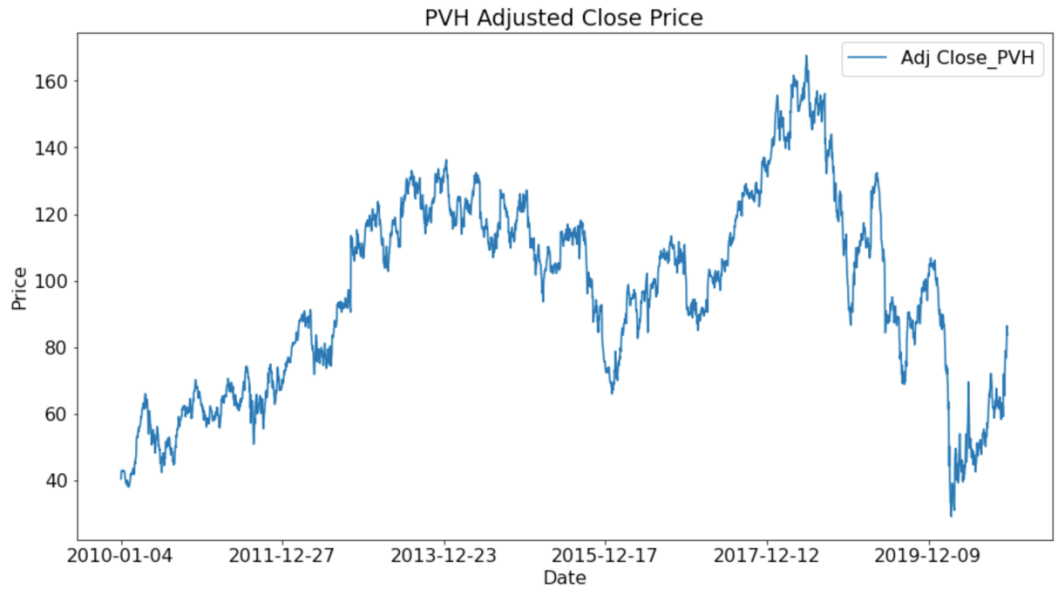
In this project, I have selected PVH stock's adjusted close price as the target. The model can be applied to any of the other stocks in S&P500. The below table shows the summary statistics for the adjusted close price of PVH.

	Adj Close_PVH					
max	**mean**	**median**	**min**	**skew**	**std**	**var**
167.603027	96.499925	99.861919	29.049999	-0.075885	29.13033	848.576102

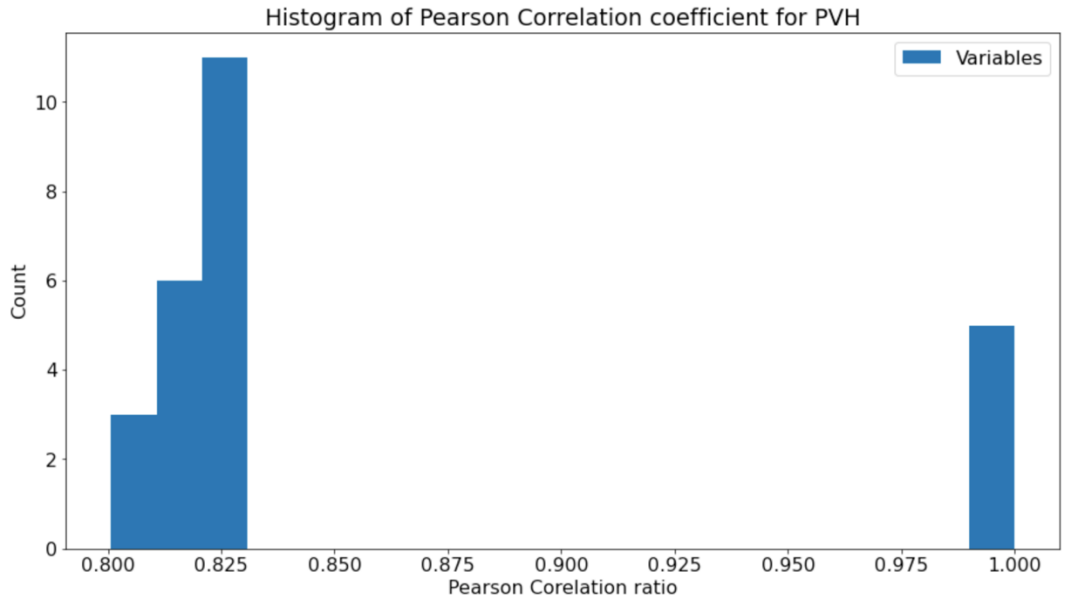
We can see that the median and mean close to each other, indicating that the target value is not skewed, having a value of just -0.07. However, the stock ranges from 29 to 167, meaning it has been relatively volatile.

Exploratory Visualization

I will visualize the adjusted close price over time for PVH stock. In the below figure, we can see an upward trend in the data until 2017, and there has been a steady decline after 2017. There is also a sharp decline in early 2020 due to covid-19. Then there has been some upward trend since late 2020.



A histogram is visualized in the below figure, depicting the Pearson correlation coefficient ratios and some variables with a greater than 0.80. A higher absolute correlation coefficient signifies that the variable can provide more information about how a specific target variable moves. For PVH adjusted close variable; there are 24 other variables with a good Pearson ratio. We would be selecting these variables as features for model building.



Algorithms and Techniques

In this project, I will be using a recurrent neural network to predict the adjusted close prices. Long-short term memory network model will be used. The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. The series of observations must be transformed into multiple examples from which the LSTM can learn. Based on the performance of the model, the sequence length would be determined.
For the selected stock target variable (Adjusted Close), all the variables with a correlation coefficient greater than 0.80 would be chosen as input features. I plan to build three LSTM Models

i. Vanilla LSTM is an LSTM model with a single hidden layer of LSTM units and an output layer used to make a prediction.
ii. Bidirectional LSTM is an LSTM model that allows the LSTM model to learn the input sequence both forward and backward and concatenate both interpretations.
iii. Stacked LSTM is an LSTM model where multiple hidden LSTM layers can be stacked on top of another.

The prediction will be one-step-ahead of the adjusted close price of the target stock. After the model is trained, the user can choose the input period, and the model would predict the next adjusted close price in the sequence

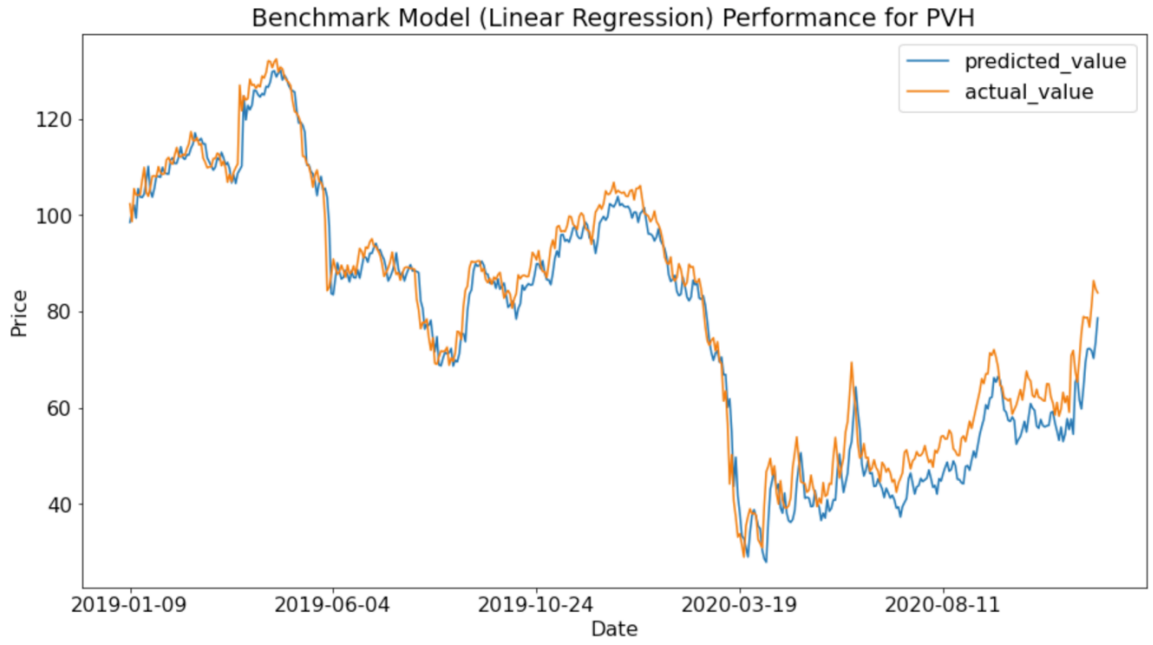
Benchmark

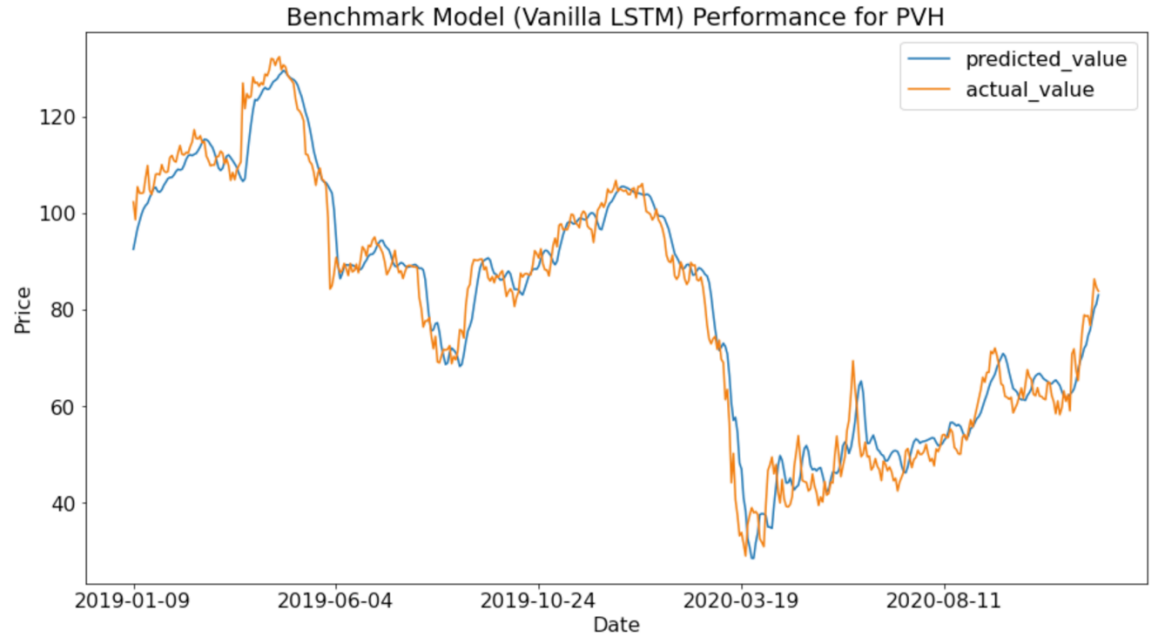
There are two benchmark models built. First, a linear regression model is built, including lagged features of the dependent variable and other exogenous features. Second a Vanilla LSTM model with one hidden layer

These benchmark models will use the lag 1 term of identified correlated variables as input to predict the current period adjusted close price. The models are trained using data from 2010 – 2018 and tested using untrained data from 2018 to 2020.

The performance documented in the below table and graphs

Stock	Model	R-Squared	RMSE
PVH	Benchmark - Regression	0.959	5.182
PVH	Benchmark - Vanilla LSTM	0.960	5.098





Methodology

Data Preprocessing

We identify the closely correlated variables to the target variable (Adjusted close price). For this, we use the Pearson corr functionality of pandas. We select all the features having a ratio greater than 0.80. We drop the target variable from this dataframe.

Next, the features are shifted ahead by 1 day. As features from the prior period are used for predicting the adjusted close for the current period. For example, features of 2020-01-01 are used for predicting the price of 2020-01-02. We turn the corresponding period features into lagged features. Each of the features (Open, Low, High, Close, Volume) are on different scales. It is necessary to normalize all of them. For this, we would use Min-Max scaler to scale all the features to range from 0 to 1.

Then we split the features into the training and testing set. Data before 2018-12-31is used for training, and data after 2018-12-31 is considered for testing.

The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. As such, the series of observations must be transformed into multiple examples from which the LSTM can learn.

An LSTM model needs sufficient context to learn a mapping from an input sequence to an output value. LSTMs can support parallel input time series as separate variables or features. Therefore, we need to split the data into samples, maintaining observations across the input sequences.

We can divide the sequence into multiple input/output patterns called samples, where x time steps are used as input, and a one-time step is used as output for the one-step prediction that is being learned.

The split_sequence() function implements this behavior and will split a given dataframe into multiple samples where each sample has a specified number of time steps. The split_labels() function breaks the target variable into a single time step. The shape of each sample input is specified in the *input shape* argument on the definition of the first hidden layer.

We almost always have multiple samples; therefore, the model will expect the input component of training data to have the dimensions or shape:

[samples, timesteps, features]

I have tried multiple timesteps length during model building and have identified 5 to yield the best results. Hence I have considered 5 as the sequence length. I use 5 days as the rolling window and move along the time series and record the data for each step as one row of the input sequence. Finally, the 2D dataframe will be converted into 3-dimensions based on the above format. Where the first dimension denotes the samples, which in our case for training is 2258, 2nd dimension is the timestep, which in our case is 5, and the 3rd dimension is the number of features, which in our case for the PVH model is 24. We will also set the rolling window on the target sequence, which is equivalent to cutting the first 5 days from the 1-dimensional array. These functions are applied to both the test and train data sets.

Implementation

All the neural network models will be implemented using the Keras module.

Vanilla LSTM Model:

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units and an output layer used to make a prediction. The shape of the input for each sample is specified in the *input_shape* argument on the definition of the first hidden layer.
We define the model with 50 LSTM units in the hidden layer and an output layer that predicts a single numerical value. We use the Relu activation function as it can solve the vanishing gradient issue.
The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or '*mse* 'loss function.
The batch size used during the training process is 1000, and the complete training process will take 500 epochs. A validation set of ~225 samples will be created randomly from the training data. By tracking validation loss, we can prevent over-fitting.

Stacked LSTM model

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model. An LSTM layer requires a three-dimensional input, and LSTMs, by default, will produce a two-dimensional output as an interpretation from the end of the sequence. We can address this by having the LSTM output a value for each time step in the input data by setting the *return_sequences=True* argument on the layer. This allows us to have 3D output from the hidden LSTM layer as input to the next.
We define the network, which takes the 3D array as input. Then there are three LSTM layers having 150, 85, and 75 hidden units, respectively. There are dropout layers after each LSTM layer to regularize the network in order to prevent over-fitting. We use the Relu activation function as it can solve the vanishing gradient issue. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or 'mse 'loss function. The batch size used during the training process is 1000, and the complete training process will take 500 epochs. A validation set of ~225 samples will be created randomly from the training data. By tracking validation loss, we can prevent over-fitting.

Bidirectional LSTM Model:

It can be beneficial to allow the LSTM model to learn the input sequence, both forward and backward, and concatenate both interpretations. This is called a Bidirectional LSTM. We can implement a Bidirectional LSTM for forecasting by wrapping the first hidden layer in a wrapper layer called Bidirectional.
We define the network, which takes the 3D array as input. There is one bidirectional LSTM layer with 150 units. Then there are two LSTM layers having 85 and 75 hidden units, respectively. There are dropout layers after each LSTM layer to regularize the network in order to prevent over-fitting. We use the Relu activation function as it can solve the vanishing gradient issue. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or 'mse 'loss function. The batch size used during the training process is 1000, and the complete training process will take 500 epochs. A validation set of ~225 samples will be created randomly from the training data. By tracking validation loss, we can prevent over-fitting.

Refinement

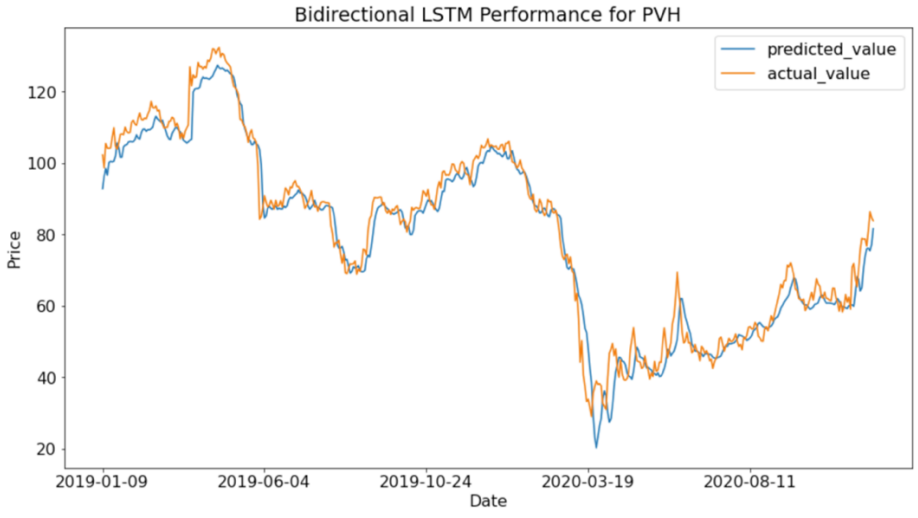
While building the various model, I tried several combinations of hyper-parameters to get the best performance.

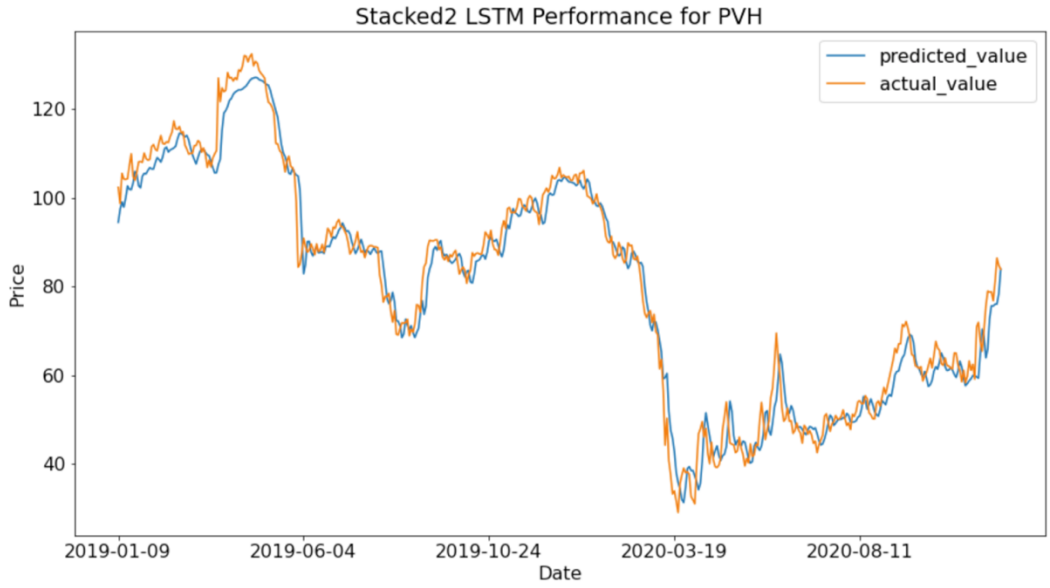
1. While calculating the correlation of the variables which need to be used as features for model development, initially, the top 200 features were considered, which produced really bad performance in the vanilla LSTM and other LSTM models. After several iterations, finalized the criteria to be greater than 0.80 correlation coefficient.
2. Timesteps required several iterations to identify the correct sequence length. Initially tried with one month of the trading window, which is 20 days. Ultimately based on the performance of the model, finalized the timestep to be 5 days, which is equivalent to 1 week.
3. I tried both mean absolute error and mean squared error as loss function, and it turns out that the model with mean squared error loss function provides better results
4. For the network structure, several combinations were tried, and finally few of the networks finalized
5. I tried using Adam and Nesterov Adam as the optimizer. The results were pretty much the same. I have used adam.
6. I tried epochs from 100 up to 1000. 1000 produced the best results and have used it.
7. Built stacked LSTM by adding an additional hidden layer to improve over the vanilla LSTM

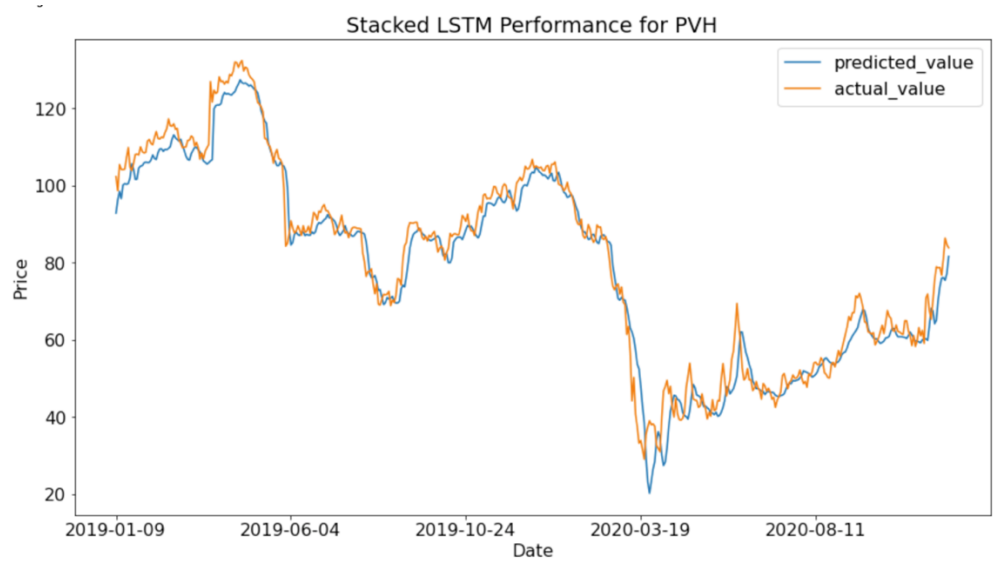
Results

Model Evaluation and Validation

The models built above using the training data can be used for predicting the predictions.
The results for different models are depicted below in the diagrams.
We can see that the prediction is quite good in all the models. For the most part, the prediction almost perfectly matches the actual value. Both R-Square and RMSE improves over the benchmark model. Although R-square is not a very high number in common sense but considering several factors in stock price determination, this result is pretty good. The RMSE is way smaller than the mean value of the testing data set, which also indicates this model has a good performance.

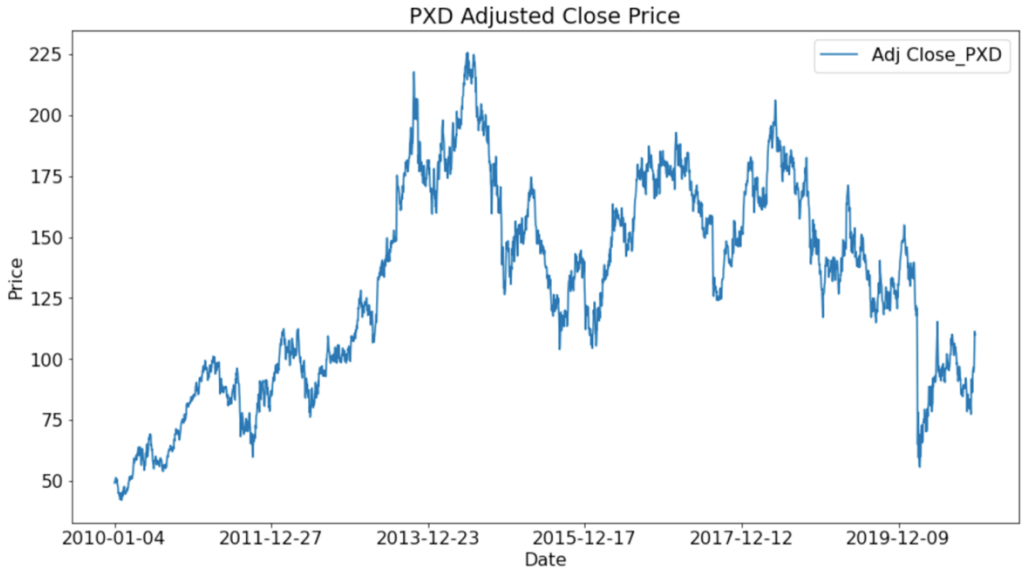


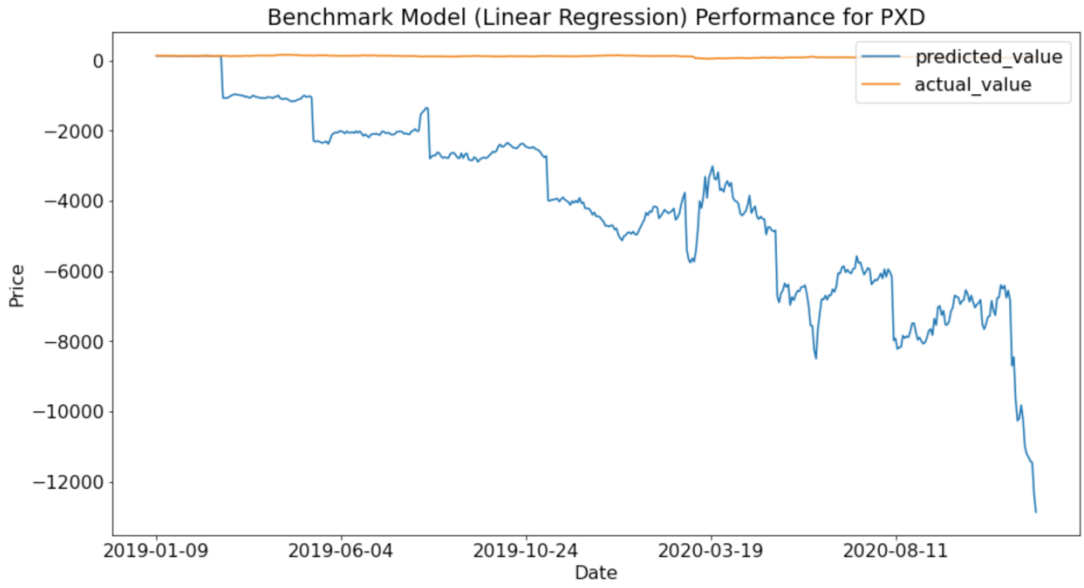


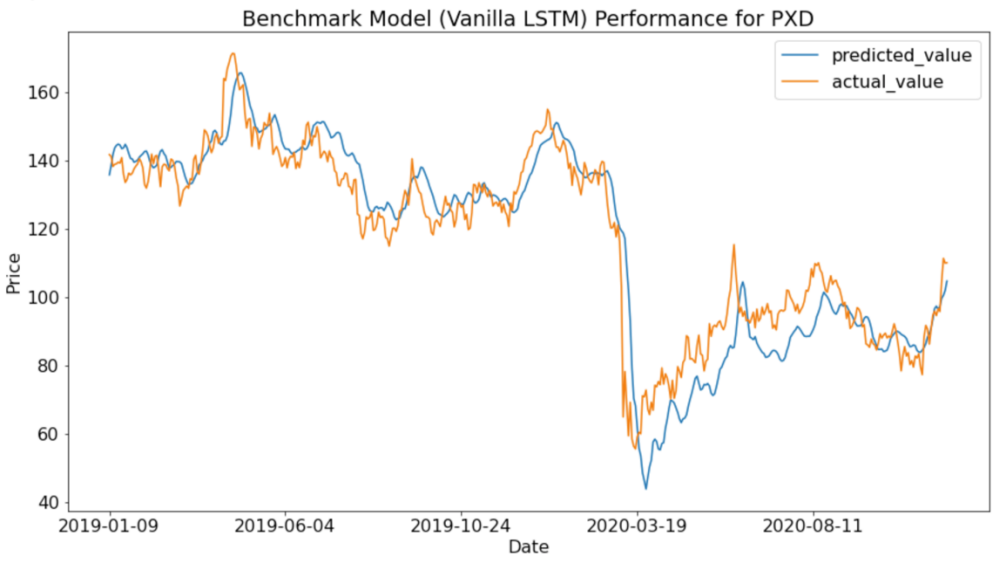


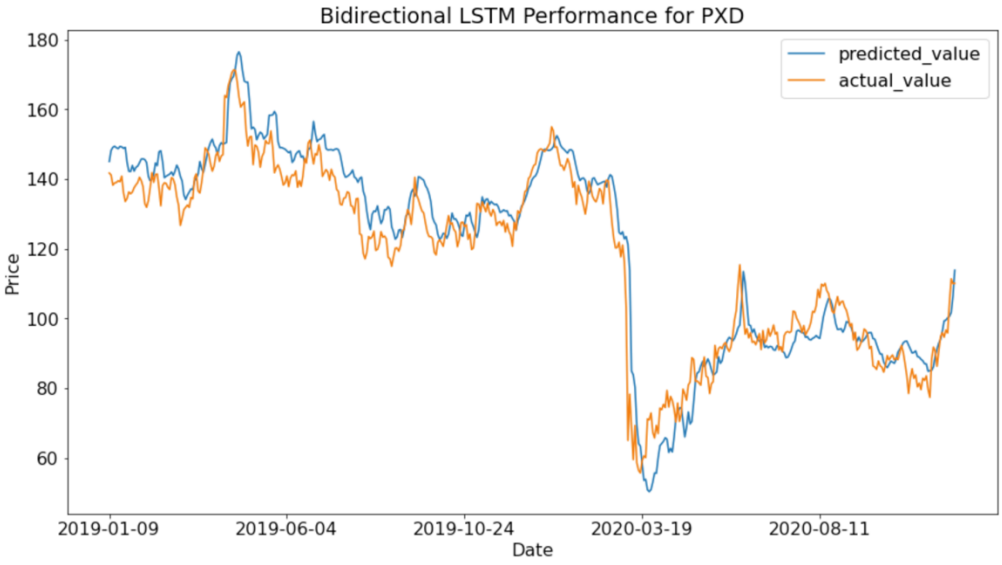
As we have a valid model built for one stock, we can try to validate the robustness of this model and check its performance on other stocks. I randomly selected another stock, PDX, and applied the same data preprocessing and LSTM models on the adjusted close price of PDX. The result is shown in the below figures and table. We can see that LSTM models perform much better than the benchmark regression model. The prediction has a similar curve as the actual value, while the predictions are shifted down a little bit. Thus, the results are not as good as PVH. However, the RMSE is way smaller than the mean value in the testing data, and considering the model is correctly predicting the upward and downward trends in the actual value, we can conclude that the model is robust with different stocks.

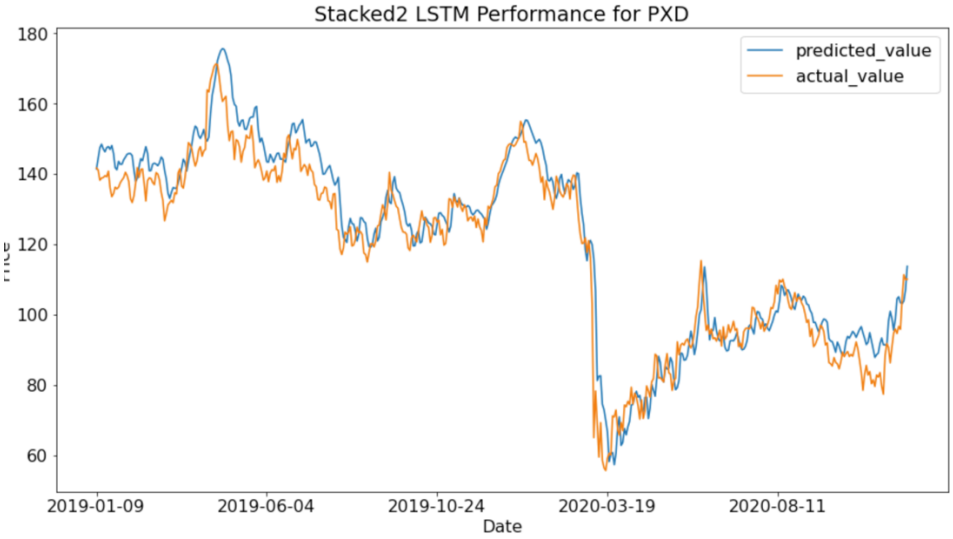
Stock	Model	R-Squared	RMSE
PXD	Benchmark - Regression	-35723.29	4848.6
PXD	Benchmark - Vanilla LSTM	0.862	9.529
PXD	LSTM-Bidirectional	0.904	7.935
PXD	LSTM-Stacked2	0.934	6.558
PXD	LSTM-Stacked	0.853	9.816

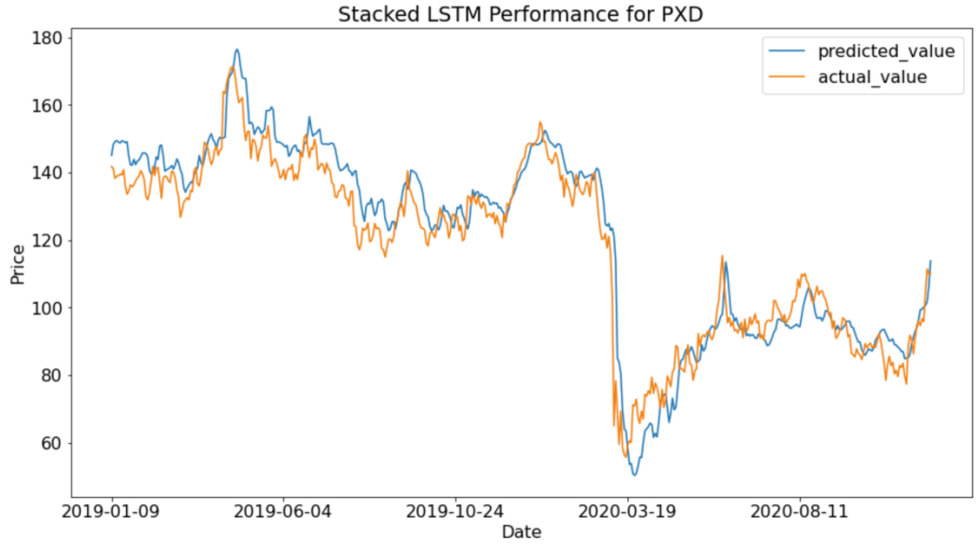

PXD Adjusted Close Price
Adj Close_PXD
Price
Date
225
200
175
150
125
100
75
50
2010-01-04
2011-12-27
2013-12-23
2015-12-17
2017-12-12
2019-12-09


Benchmark Model (Linear Regression) Performance for PXD
predicted_value
actual_value
Price
Date
0
−2000
−4000
−6000
−8000
−10000
−12000
2019-01-09
2019-06-04
2019-10-24
2020-03-19
2020-08-11


Benchmark Model (Vanilla LSTM) Performance for PXD
predicted_value
actual_value
Price
Date
160
140
120
100
80
60
40
2019-01-09
2019-06-04
2019-10-24
2020-03-19
2020-08-11


Bidirectional LSTM Performance for PXD
predicted_value
actual_value
Price
Date
2019-01-09
2019-06-04
2019-10-24
2020-03-19
2020-08-11


Stacked2 LSTM Performance for PXD
predicted_value
actual_value
Date
2019-01-09
2019-06-04
2019-10-24
2020-03-19
2020-08-11


Stacked LSTM Performance for PXD
predicted_value
actual_value
Price
Date
2019-01-09
2019-06-04
2019-10-24
2020-03-19
2020-08-11

Justification

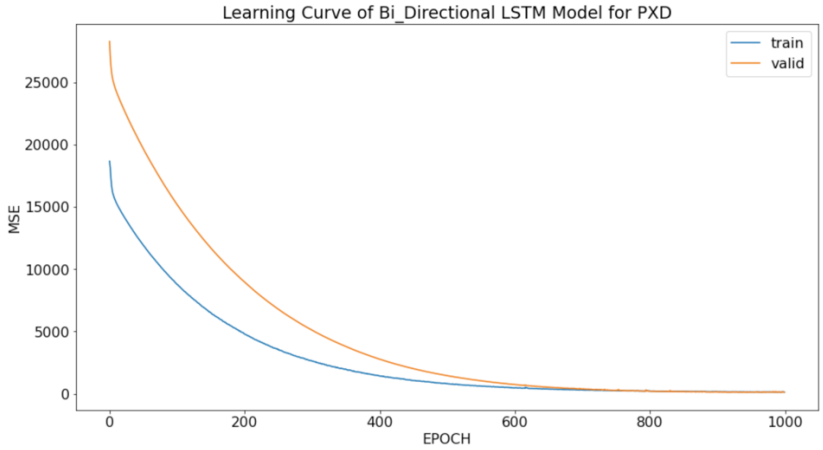
The below table depicts the model performance comparison between benchmark models and LSTM models for two stocks PVH and PXD. We can clearly see that the Stacked and bi-directional LSTM models perform much better than the benchmark regression models. LSTM models generated its own feature representations in the hidden units, and its long-short term memory cell is good at saving useful information and discarding useless information through its gate units. Hence LSTM is simple and useful in dealing with time-series data. Although the metrics depict that the LSTM models are not completely perfect, but they are still good at predicting stock prices.

Stock	Model	R-Squared	RMSE
PVH	Benchmark - Regression	0.9592	5.182
PVH	Benchmark - Vanilla LSTM	0.929	6.83
PVH	LSTM-Bidirectional	0.959	5.17
PVH	LSTM-Stacked2	0.969	4.461
PVH	LSTM-Stacked	0.889	8.513
PXD	Benchmark - Regression	-35723.29	4848.6
PXD	Benchmark - Vanilla LSTM	0.862	9.529
PXD	LSTM-Bidirectional	0.904	7.935
PXD	LSTM-Stacked2	0.934	6.558
PXD	LSTM-Stacked	0.853	9.816

Conclusion

Free-Form Visualization

An important aspect of neural networks is the convergence of its loss function. Convergence will show that a model is successfully trained and if there could be an under-fitting or over-fitting issue. Below visualization shows the learning curve of the training process during the bi-directional model building. We observe the curves of both the loss functions for training and validation set is decreasing as the number of epochs increases and converges to almost zero. This indicates that the model is successfully training and is converging into a local minimum. The validation loss decreases and becomes less volatile as the number of epoch increases indicate there is no over-fitting or under-fitting problem in the training process.



Reflection

This project involves building a machine learning model to prepare the adjusted close price of a specific stock using the historical features (Low, close, high and volume) of itself and other closely correlated stocks. Historical data for the last 10 years is downloaded from yahoo finance using yfinance. Getting this data was difficult as developer API access to yahoo finance was shut down after May 2017; it took a bit of effort and research to get free access to high-quality stock data. Finally found yfinance, which provided quality historical data.
Next step, I use the Pearson correlation coefficient for feature selection. I set the threshold as 0.80 to select all the features which need to be shortlisted for model training. Depending on the target we are trying to predict, the number of features is determined. For example, for PVH, we get 24 features, and for PXD, we get 54.
Next, the data is normalized using a min-max scaler to make all the features on the same scale. Then we calculate the time steps. We have selected the sequence length of 5 days. This creates a rolling window of 5 days' worth of features, which would be used for predicting the adjust the close price of the stock for the next day.
We split the data into test and train set to keep the training and testing data-independent and not cause information leakage.
We build and test the benchmark regression and Vanilla LSTM models and document their performance. Next, we build the stacked LSTM and Bi-Directional by identifying the various hyper-parameters for the network and model. All the LSTM models are trained for 1000 epochs. I will use r squared and RMSE for evaluating the performance of all the models. The final result shows that the LSTM Stacked model is better than the benchmark model and can successfully and robustly predict the adjusted close price of multiple stocks. I think this model could be implemented on other stocks and could yield a good prediction. An interesting aspect of the project was that the stacked LSTM model consistently performed better than the bi-directional LSTM model.

Improvement

We could improve the data used for this project. Rather than just considering the daily trading data, we could potentially supplement it with fundamental data like revenues, market cap, profit etc., and try to include the market sentiment data by scrapping Twitter and other social media. This will enrich the data available for model building and improve the performance and make it more generic.
We could also improve on the feature selection process by incorporating PCA methods. Also, use a larger compute capability to build more complex networks or train for larger epochs. I believe the model's performance could be further improved if we include the improvements as mentioned above.

Reference

1. Das, Shom Prasad, and Sudarsan Padhy. "Support vector machines for prediction of futures prices in Indian stock market." International Journal of Computer Applications 41-3-2012
2. Bao, Wei, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." PloS one 12-7-2017
3. Datacamp.com
4. machinelearningmastery.com