Machine Learning Engineer Nanodegree

Capstone Report

Liangliang Yang

Part (I). Definition

1.1 Project Overview

Time series forecasting, especially stock prediction, has been a hot topic for decades. Obviously, predicting the stock market is one of the most challenging things to do, and so many smart people and organizations are involved in this area. There are many variables that will affect the price [1]: the earnings of a company, the supply and demand at that time, the trends of the overall economy, the political climate and so on.

A screen showing the stock prices at the New York Stock Exchange[2]

Traditionally, people built many stock indicators, and applied them to rule based models and to predict whether a stock will go up or down in the future. In recent years, as machine learning and deep learning become more and more popular, people start to use various new techniques to build data-driven models. With the power of big data and AI, people now have the potential to achieve faster and more accurate predictions.

1.2 Problem Statement

In this project, I will create a stock predictor with machine learning algorithms. In Particular, I will use a deep learning technique named “Long Short Term Memory (LSTM), which is a model that extends the memory of recurrent neural networks. The model will mainly focus on the prediction of “Adj Close” price for next 7 business days, based on historical data. A benchmark model will be provided, and evaluation analysis will be performed to test the model performance.

When the LSTM model building is done, the second main piece of this project is an interactive web application. It will enable users to pick different stock symbols and time ranges for model running, and see the price forecasting of the next 7 days. For this part, I plan to use Dash and Plotly [3]. It is a Python framework for building analytic web applications

1.3 Metrics

For the metrics of model performance, evaluation, I will mainly use Root Mean Square Deviation (RMSD) [4] and Mean Absolute Percentage Error (MAPE).

Root Mean Square Deviation (RMSD), sometimes known as Root Mean Square Error (RMSE), is the square root of mean squared error (MSE). Below is the equation for RMSD.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}}$$

In the equation, the $\widehat{y_t}$ is the prediction for Adjusted Close price, and y_t is the real Adjusted Close price for the stock at time t.

In addition to the Root Mean Square Deviation, I will also use Mean Absolute Percentage Error (MAPE). It is another measure of prediction accuracy. In our project, we expect the MAPE is within 5% for 7 days forecasting for a good model. The equation for MAPE is as below.

$$MAPE = \frac{1}{T} \times \left(\sum_{t=1}^{T} \frac{|\widehat{y_t} - y_t|}{y_t}\right) \times 100\%$$

Part (II). Analysis

2.1 Data Exploration

For this project, I will use data from Yahoo Finance. I will use a python package named yfinance [5], which can help download the stock data with different symbol and date ranges.

Here I have created a function to download stock data and convert it to a dataframe, which can be further used in data exploration analysis and modeling.

```
def download_stock_to_df(symbol, start, end):
    """
    Get current stocks data from yahoo fiance and save to dataframe

    Params:
        symbol: stock to pull data
        start: start date of pulled data
        end: end date of pulled data

    Return:
        dataframe of stock within specified date range
    """
    df_stock=yf.download(symbol,start,end,progress=False)
    df_stock.reset_index(level=0, inplace=True)
    return df_stock
```

In the project, I will create a web application (with Dash Plotly), which will enable the users to pick different stock symbols and date ranges for model training and forecasting. On the other hand, for the purpose of data exploration and model building analysis, I will focus on the Apple stock prices (AAPL). It is a famous company and it also has a long history, so the data size will be better. Personally I also have great interest in this stock.

I will pull data from ‘2000-01-01’ to ‘2020-05-31’, and use this 20 years of stock data as our training and testing. An overview of the stock DataFrame is as below.

```
apple_stock = download_stock_to_df('AAPL', '2000-01-01', '2020-05-31')
apple_stock.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2000-01-03	3.745536	4.017857	3.631696	3.997768	3.460857	133949200
1	2000-01-04	3.866071	3.950893	3.613839	3.660714	3.169071	128094400
2	2000-01-05	3.705357	3.948661	3.678571	3.714286	3.215448	194580400
3	2000-01-06	3.790179	3.821429	3.392857	3.392857	2.937188	191993200
4	2000-01-07	3.446429	3.607143	3.410714	3.553571	3.076317	115183600

As we can see from the above table, the DataFrame contains 7 columns. For this project, our main goal is to predict the Adj Close price.

We can use the 'describe' method of pandas to get more information about the data.

```
print('Total data points: {}'.format(len(apple_stock)))
print('From {} to {}'.format(apple_stock['Date'].iloc[0], apple_stock['Date'].iloc[-1]))
```

```
Total data points: 5134
From 2000-01-03 00:00:00 to 2020-05-29 00:00:00
```

```
apple_stock.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	5134.000000	5134.000000	5134.000000	5134.000000	5134.000000	5.134000e+03
mean	64.154592	64.803741	63.514898	64.184123	60.260955	1.135283e+08
std	72.004205	72.762315	71.358150	72.106102	70.781017	9.831566e+07
min	0.927857	0.942143	0.908571	0.937143	0.811282	9.835000e+06
25%	5.160000	5.238929	5.068572	5.137500	4.447520	4.634312e+07
50%	32.162857	32.317856	31.890714	32.167143	27.847013	8.562260e+07
75%	104.972500	106.052500	104.485003	105.250002	97.635321	1.501342e+08
max	324.739990	327.850006	323.350006	327.200012	326.316681	1.855410e+09

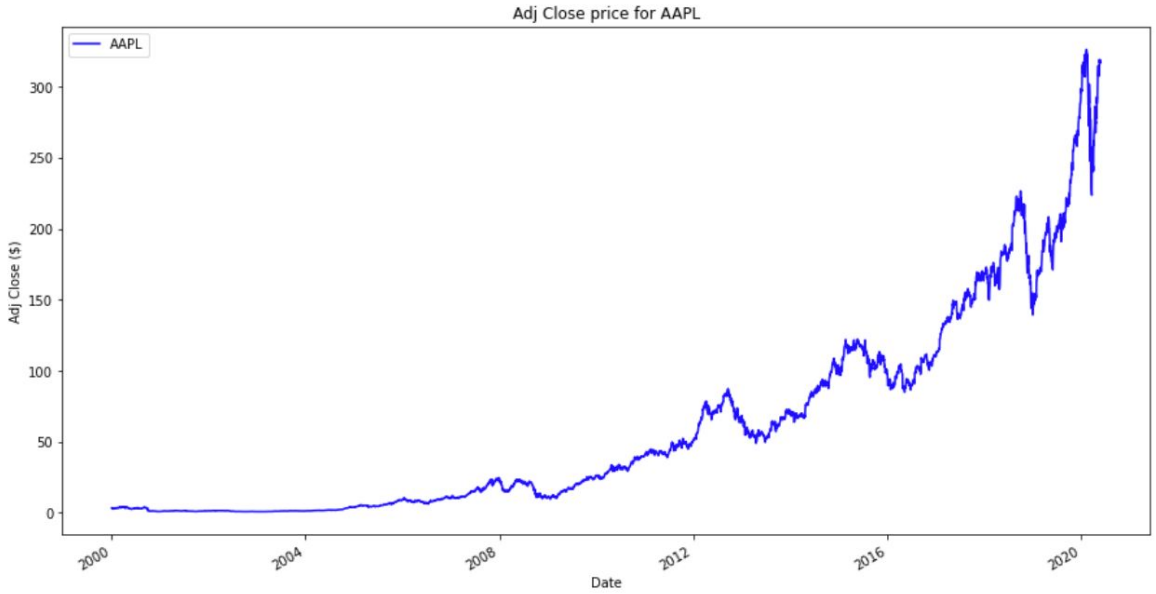
As we can see above, the data contains 5134 points, and we will later split them into train and test datasets. The table also tells a lot of statistical information about the AAPL stock data. For example, we can see the minimum 'Adj Close' price is about $0.81, and the maximum price is about $326.3. That is about 400 times the difference. Really hope I can buy some AAPL stock 20 years ago.

2.2 Exploratory Visualization

To better understand the stock data, we can use matplotlib to perform some exploratory visualization analysis.

We can first take a look at the ‘Adj Close’ for the last 20 years.

```
ax = apple_stock.plot(x='Date', y='Adj Close', title='Adj Close price for AAPL', figsize=(15, 8), color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Adj Close ($)')
ax.legend(['AAPL'])
plt.show()
```

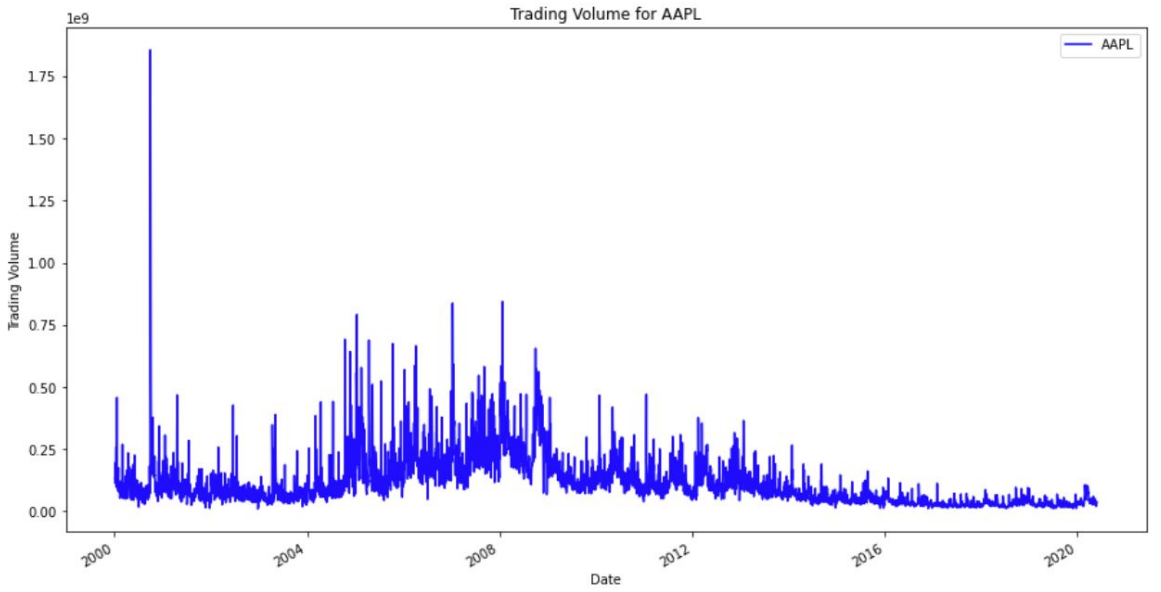


As we can see above, there are four peaks occurred after 2010: (1) 2012/07-2012/09; (2)2015/04; (3)2018/09; (4) 2020/02. The most recent big drop and up , which happened this Spring, is still a fresh memory for us all.

Since there is a big change this Spring for AAPL (and for most other stocks as well), I plan to include this time period in the test dataset. This would be challenging for the model.

In addition to the price, let’s also take a look at the stock volume.

```
ax = apple_stock.plot(x='Date', y='Volume', title='Trading Volume for AAPL', figsize=(15, 8), color='blue')
ax.set_xlabel('Date')
ax.set_ylabel('Trading Volume')
ax.legend(['AAPL'])
plt.show()
```



Interestingly, as we can see from the plot, there is a big peak at the beginning. Let's find it out.

```
apple_stock[apple_stock['Volume'] == apple_stock['Volume'].max()]
```

	Date	Open	High	Low	Close	Adj Close	Volume
188	2000-09-29	2.013393	2.071429	1.8125	1.839286	1.592265	1855410200

The volume peak happened on '2000-09-29', and if we search on Google, we can see that it is "Apple's Worst Day Ever". On September 29th, 2000, AAPL saw a massive one-day drop of 51.89%. The stock's steep decline was due to a number of factors, which in hindsight combined to form a perfect storm of headwinds that proved too fierce for shareholders to deal with at the time. [News link]

2.3 Algorithm and Techniques

After data exploration and visualization, now we have a better understanding of the data. Next we will focus on two steps: (1) build a stock prediction model (2) build a web application for stock predictor.

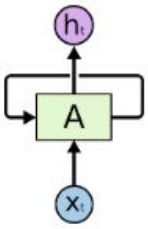
For the prediction model, we will mainly focus on the LSTM model, and then use the Dash to create a simple web application for it.

2.3.1 Algorithm - LSTM

Long Short Term Memory (LSTM), which was first introduced by Hochreiter & Schmidhuber in 1997 [6], is an artificial recurrent neural network (RNN) model or architecture.

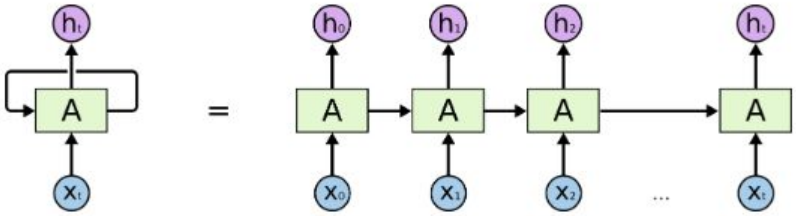
LSTMs are very powerful in sequence (time series) prediction problems (such as stock price prediction in our case), as they're able to store past information.

To better understand the LSTM network, we need to explain the RNN. A typical RNN looks like below:[7]



In the above diagram, A is a chunk of neural network. The x_t is the input (in our case, historical stock price information), and the h_t is the output (the stock price in the future).

It will be easier to understand the process if we unfold the above neural network. [7]

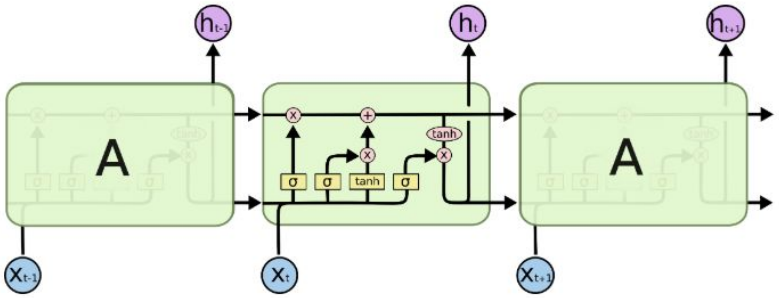


An unrolled recurrent neural network.

As we can see from the above graph, the prediction at time t (which is h_t), is dependent on all previous predictions and information.

The problem with RNNs is that they don't know which information is more important, which is not that important. They can learn and remember things for small durations of time, and make simple predictions. However, when the problem gets more complicated, and more past information will be needed, the RNNs can't learn very well.

LSTMs on the other hand, make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. A simple diagram of LSTMs is as below.[7]



The repeating module in an LSTM contains four interacting layers.

The LSTMs mainly contain three gates:

- The input gate: The input gate adds information to the cell state
- The forget gate: It removes the information that is no longer required by the model
- The output gate: Output Gate at LSTM selects the information to be shown as output

With the above three gates, the LSTMs are able to store past information that is important, and forget those non-important, which creates the power to remember and learn information for long periods.

2.3.2 Technique - Dash

Most data analysis and model building processes were run with Jupyter Notebook. Apart from that, a technique named Dash [8] is also used to create a web application. The Dash is built on top of Plotly.js, React and Flask, Dash ties modern UI elements like dropdowns, sliders, and graphs directly to your analytical Python code. The Dash app code is declarative and reactive, which makes it easy to build complex apps that contain many interactive elements.

2.4 Benchmark

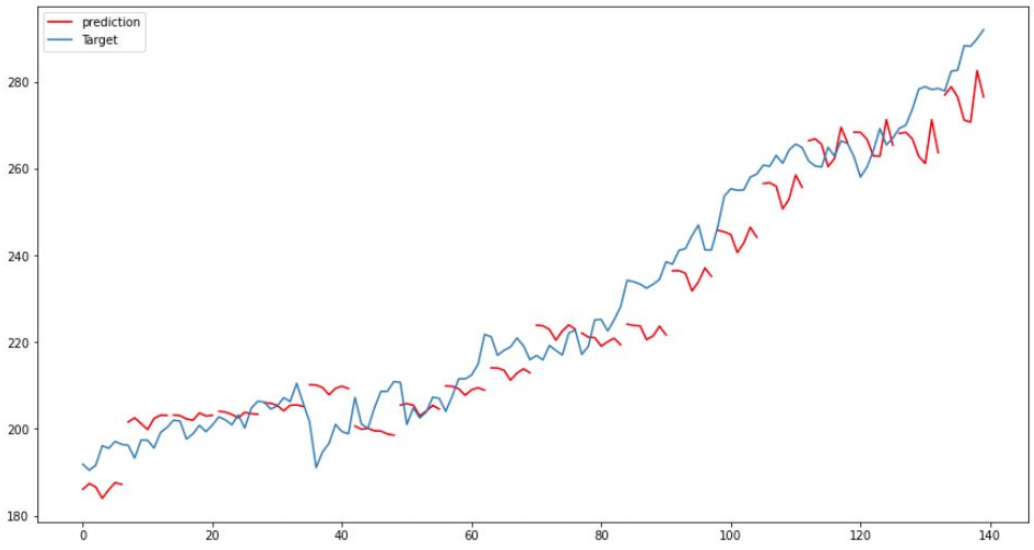
The benchmark model that I will use in the project is Simple Moving Average [9]. A moving average (MA) is a widely used indicator in technical analysis that helps smooth out price action by filtering out the “noise” from random short-term price fluctuations.

For example, if we want predict next one day stock price based on last N days of dats, then the equation will be: [9]

$$\begin{aligned}\bar{p}_{\mathrm{SM}} &= \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n} \\ &= \frac{1}{n}\sum_{i=0}^{n-1} p_{M-i}.\end{aligned}$$

And if we want to predict/calculate prices for more than one day, then we will need to drop the oldest data point, and add the prediction value to the data queue.

In the project, for the purpose of comparison, I will train and predict the stock price for the same time range. If we use 7 days prediction as our forecasting window, then I will use k (eg, 20) individual continuous windows for the model comparison and evaluation. For example, if we train the model with data from day 1 to day 1000, then I will test the performance with windows [(1001, 1007), (1008, 1014), … (1134, 1140)], and then compare the predictions to the real prices. A simple prediction plot example is as below:



Part (III). Methodology

3.1 Data Preprocessing

Since we only need to predict the 'Adj Close' price for the stock, I will just use the 'Adj Close' column for our data.

```
# keep only used column
apple_stock.drop(columns=['Open', 'High', 'Low', 'Close', 'Volume'], inplace=True)
```

For neural network models (including LSTM), when we fit them on data that is not scaled, it is possible to slow down the learning and convergence. Hence, we need to normalize the data before model training.

```
sc = MinMaxScaler(feature_range=(0.01, 0.99))
df['Scaled_Close'] = sc.fit_transform(df[['Adj Close']])
```

In addition, we need to split the data into train and test datasets. Here we define 4 main parameters as below.

```
num_periods = 40
forcast_days = 7
lookback_days = 15
window_size  = 15
```

The ‘num_periods’ is prediction periods that we will test, and compare the results to the benchmark model. The ‘forcast_days’ is defined as how manys we want to forecast based on historical data. The ‘lookback_days’ will be used for the LSTM model as the input feature shape, and the ‘window_size’ is the parameter for the Simple Moving Average benchmark model.

We can then split the data as below. The result train and test sets will have three columns. Beside, for the test dataset, we will need to create an extended dataset, so that it will contain some previous prices, for the purpose of feature creation and moving average price calculation.

```
N = df.shape[0]
N_test = num_periods*forcast_days
N_train = N - N_test
```

```
from sklearn.model_selection import train_test_split
dataset_train , dataset_test = train_test_split(df, train_size=N_train, test_size=N_test, shuffle=False)
```

```
dataset_test_extend = dataset_train[-lookback_days:].append(dataset_test)
print(dataset_train.shape, dataset_test.shape, dataset_test_extend.shape)
```

```
(4854, 3) (280, 3) (295, 3)
```

```
dataset_test.head()
```

	Date	Adj Close	Scaled_Close
4854	2019-04-22	201.342468	0.613740
4855	2019-04-23	204.246475	0.622483
4856	2019-04-24	203.931473	0.621535
4857	2019-04-25	202.080780	0.615963
4858	2019-04-26	201.116043	0.613058

3.2 Implementation

Once we have defined our parameters such as ‘lookback_days’, ‘forcast_days’, and have our train and test datasets. The next step is to generate the sequence for the LSTM model. Below we have the function to do that.

```
# train sequence is continuous
def generate_train_sequence(train_data, lookback_days, forcast_days):
    """
    Generate sequence array for train data,
    inclduing both X and y
    """
    X = []
    y = []
    m = lookback_days
    n =forcast_days
    N = len(train_data)

    for i in range(0, N, 1):
        # input sequence : x[i].....x[i+m]
        # output sequence: x[i+m+1]....x[i+m+n]
        # last index is (i+m+n)
        end_index = i + m + n # find the end of this sequence
        # check if we are out of index
        if end_index > N-1:
            break
        seq_x = train_data[i:(i+m)]
        seq_y = train_data[(i+m):(i+m+n)]
        X.append(seq_x)
        y.append(seq_y)

    array_X = np.array(X) # shape (N, m, 1)
    array_y = np.array([list(a.ravel()) for a in np.array(y)]) # shape (N, n, 1) convert to (N, n)
    return array_X, array_y
```

We can generate our training sequence as below.

```
X_train, y_train = generate_train_sequence(train_set, lookback_days, forcast_days)
print(X_train.shape, y_train.shape)
```

```
(4832, 15, 1) (4832, 7)
```

For each X_train data point, the shape is (15, 1), which is the last 15 days of normalized ‘Adj Close’ price. The datapoint for y_train, will be an array of 7 days prices. If we want to know the real price after prediction, we will need to use the normalization function to convert it back.

Since the training data is only 4832 points, I will just use a simple LSTM model, instead of a complicated model (which will be two or more layers). The loss function of the LSTM will be `mean_squared_error’, and the activation function I will choose ‘relu’. The input data shape, as expected, will be (lookback_days, 1). The model function is as below.

```
model = Sequential()
model.add(LSTM(units=30, activation='relu', input_shape=(lookback_days,1)))
model.add(Dense(forcast_days))
model.compile(optimizer='adam', loss='mean_squared_error')
```

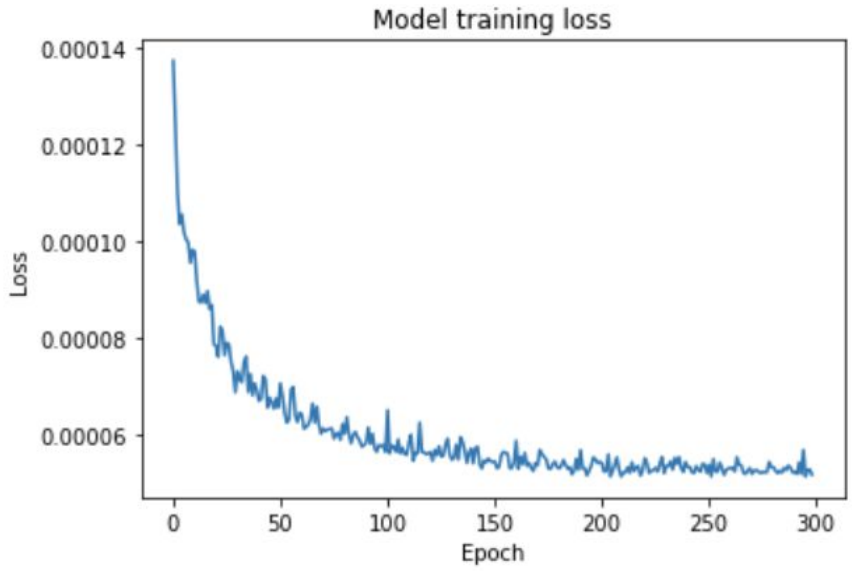
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 30)                3840
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 217
=================================================================
```

As I don't know how long it will take for the model to converge, I prefer to use a large epoches number, and do further analysis about it.

```
num_epochs = 300
history  = model.fit(X_train, y_train, epochs=num_epochs, batch_size=32)
```

It will take about 10 minutes for the model to run. Once it is done, we can plot the training loss versus the epoches as below.



As we can see above, the model reaches convergence when epoch is above 200.

Once we have finished the model training, we need to test the model prediction on the test dataset. As we mentioned in the previous sections, we will use 40 periods of 'forecast_days' to test the prediction (for 7 days forecast, this will be 280 days).

We will also need to create the test data sequence for the LSTM model input. However, unlike the training data, we will create 40 separate sequences (no overlap). Below is the code to do that.

```
def generate_test_sequence(test_data, lookback_days, forcast_days):
    """
    Generate sequence array for test data,
    inclduing only X
    """
    X = []
    y = []
    m = lookback_days
    n =forcast_days
    N = len(test_data)

    for i in range(0, N, n): # here use 7, not 1, as we will use groups
        # input sequence : x[i].....x[i+m]
        # output sequence: x[i+m+1]....x[i+m+n]
        # last index is (i+m+n)
        end_index = i + m + n # find the end of this sequence
        # check if we are out of index
        if end_index > N:
            break
#         print(i, i+m, i+m+n)
        seq_x = test_data[i:(i+m)]
        seq_y = test_data[(i+m):(i+m+n)]
        X.append(seq_x)
        y.append(seq_y)

    array_X = np.array(X) # shape (N, m, 1)
    array_y = np.array([list(a.ravel()) for a in np.array(y)]) # shape (N, n, 1) convert to (N, n)
    return array_X, array_y
```

```
X_test, y_test = generate_test_sequence(test_set_extend, lookback_days, forcast_days)
print(test_set_extend.shape, X_test.shape, y_test.shape)
```

```
(295, 1) (40, 15, 1) (40, 7)
```

Once we have the above test data sequence, we can use our trained LSTM model to perform the prediction. Note that the prediction value will be a normalized price, so we will need to use the ‘inverse_transform’ to transform it back to real price.

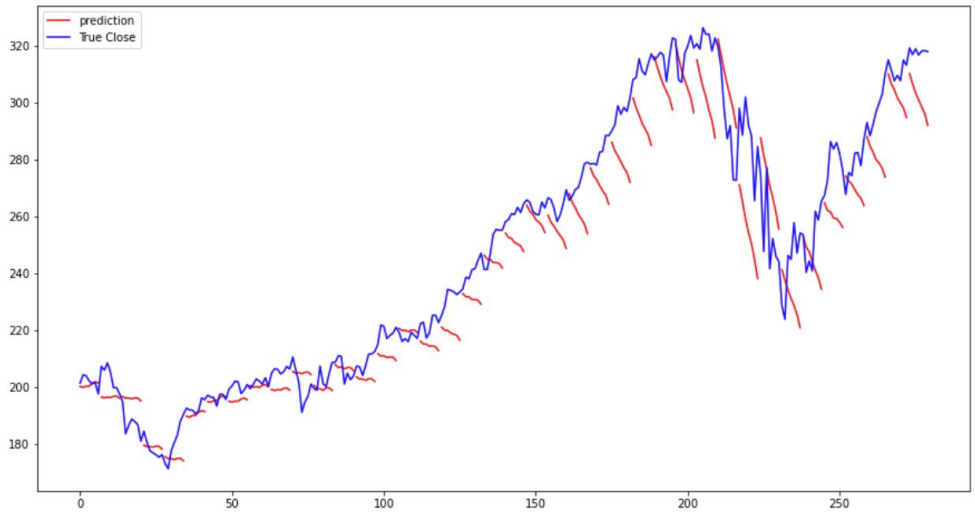
```
LSTM_prediction_scaled = model.predict(X_test)
LSTM_prediction = sc.inverse_transform(LSTM_prediction_scaled)

train_set = train_set.reshape((-1))
test_set = test_set.reshape((-1))
LSTM_prediction = LSTM_prediction.reshape((-1))
```

```
dataset_test['LSTM_Prediction'] = LSTM_prediction
dataset_test.head()
```

	Date	Adj Close	Scaled_Close	LSTM_Prediction
4854	2019-04-22	201.342468	0.613740	200.179977
4855	2019-04-23	204.246475	0.622483	199.795197
4856	2019-04-24	203.931473	0.621535	200.244537
4857	2019-04-25	202.080780	0.615963	200.233948
4858	2019-04-26	201.116043	0.613058	201.373123

We can visualize the prediction price along with the real price, with matplotlib.

```
def plot_prediction_comparison(true_set, prediction_set):
    plt.figure(figsize = (15,8))

    for i in range(0, len(prediction_set), forcast_days):
        indexs = [i+x for x in range(forcast_days)]
        values = prediction_set[i:i+forcast_days]
        plt.plot(indexs, values, color='r')

    plt.plot(0, prediction_set[0], color='r', label='prediction') # just used to add label

    plt.plot(true_set, color='b', label='True Close')
    plt.legend(loc='best')
    plt.show()
```



As we can see above, when the price increases too fast, the prediction keeps output a big drop signal, in which some of them are correct, some of them are not.

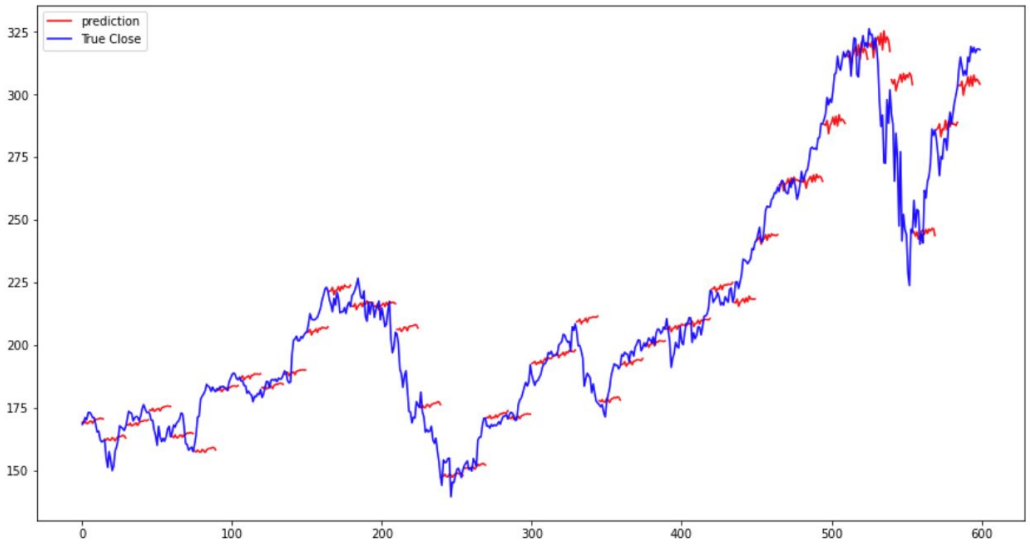
In addition to the 7 days forecast, I have also built a model for 15 days forecast, as below.

```
model = Sequential()
model.add(LSTM(units=30, activation='relu', input_shape=(lookback_days,1)))
model.add(Dense(forcast_days))
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 30)                3840
_________________________________________________________________
dense_1 (Dense)              (None, 15)                465
=================================================================
```

For the 15 days forecast, as you can see from the model, I use 30 days as look_back_days. The prediction results are as below. I will discuss more about the model performance in section (IV).



3.3 Refinement

The above model implementation is more towards a static way. It is not very convenient for the user to use, and check the price on a selected day. Hence, we are going to build a web application, which can help users train and visualize the stock in an interactivate way.

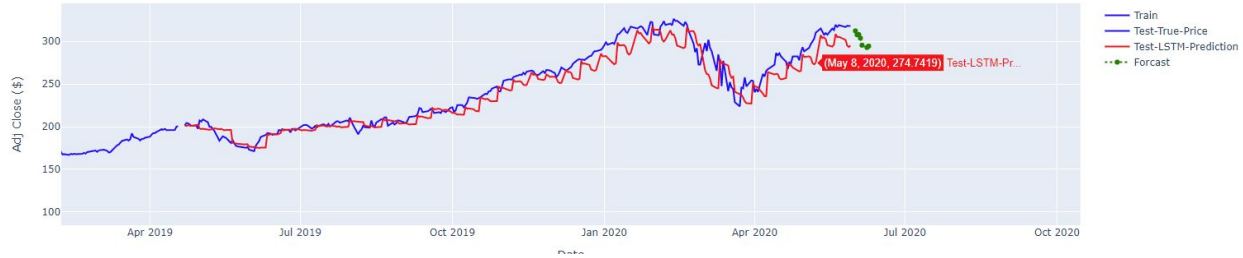
As we mentioned, the Dash Plotly will be used. The related code is in the 'lstm' folder. In order to start the web application, we will need to run *'Python index.py',* and then open the web address at 'http://127.0.0.1:8080/'.

As we see below, the application looks like below:

We can pick the stock symbol by choosing from the dropdown window, and select the date range to download data. Similar as before, the data will be split into train and test. After finishing the selection, we can run the model by clicking on the 'Run Model' button.

You can monitor the program progress from the terminal, usually it will take about 5-10 minutes. Once done, we will be able to see the result plot.

Unlike the static plot in the jupyter notebook, here the result is an interactive plot. You can view the details in each stock data point. More importantly, you can select a subset of the figure and zoom it bigger, so you can see the details. For example, we can zoom the test and prediction range for the above figure as below.



Part (IV). Results

4.1 Model Evaluation and Validation

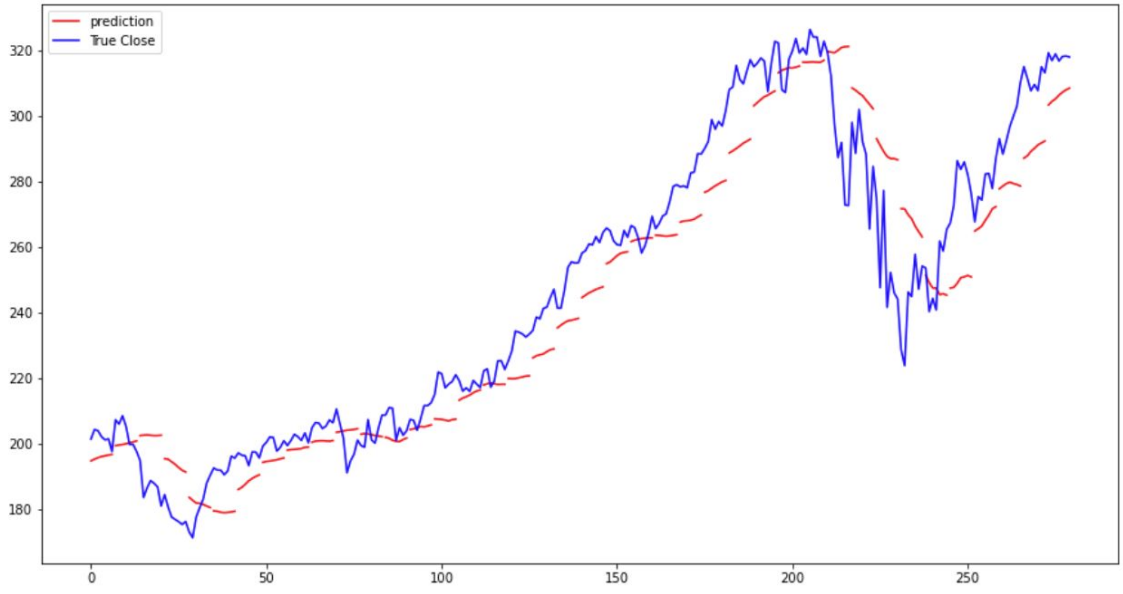
For model evaluation, I used a test dataset, which contains 40 periods (if forcast_days=7, then totally 280 days).

For the purpose of better model evaluation, let's create a function that can do moving average prediction.

For the moving average, it is easy to do one day forward calculation. How about 7 days? 15 days? Here we will use an approximation method. For example, for a 7 days prediction, we will first calculate the moving average for the 1st day, based on historical real price (say, a window of 15 days). After we get the 1st day prediction, we will append the prediction price to the historical price list, and move the window one day forward. For the 2nd day prediction, we will use 14 days of real Adj Close price and 1 day of prediction price. So on so forth. The function is as below.

```
dataset_test_extend['MA_Prediction'] = np.NaN

def make_window(size, start):
    # make index window for use of concat of real price and prediction price
    return [start+k for k in range(size)]

for index in range(window_size, len(dataset_test_extend), forcast_days):
    for i in range(0,forcast_days):
        if index+i >= len(dataset_test_extend):
            break
        # window for real price, eg [1, 2, 3, 4, 5, 6, 7, 8, 9]
        window_real = make_window(window_size-i, index+i-window_size)

        # window for predicted price, eg [10]
        window_MA = make_window(i, index)

        price_window = pd.concat([ dataset_test_extend['Adj Close'].iloc[window_real],
                                   dataset_test_extend['MA_Prediction'].iloc[window_MA] ])
        next_mean_prediction = price_window.mean(axis=0)

        dataset_test_extend.iat[index+i, dataset_test_extend.columns.get_loc('MA_Prediction')] = next_mean_prediction
```

We can then calculate the simple moving average prediction price for the test dataset, and plot the comparison of prediction versus real price.



For better visualization, we can put the two prediction prices in the same plot.

From the above plot, as we can see, for a 7 days forecast, both predictions are not very bad. To quantitatively understand the performance of the model, let's calculate the Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE).

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# metrics.mean_absolute_error(y_test, y_pred)

def evaluate_prediction(y_true, y_pred):

    # mean absolute error
    mae = mean_absolute_error(y_true, y_pred)

    #mean squared error
    mse = mean_squared_error(y_true, y_pred)

    # root mean squared error
    rmse = round(np.sqrt(mse), 2)

    # mean absolute percentage error
    mape = round(np.mean(np.abs((y_true - y_pred) / y_true)) * 100, 2)

    return {'root_mean_squared_error': rmse,
            'mean_absolute_percentage_error': mape}
```

```
y_test = dataset_test['Adj Close']
y_pred_lstm = dataset_test['LSTM_Prediction']
y_pred_ma = dataset_test['MA_Prediction']
```

```
# evaluation for LSTM prediction
evaluate_prediction(y_test, y_pred_lstm)
```

```
{'root_mean_squared_error': 13.51, 'mean_absolute_percentage_error': 3.9}
```

```
# evaluation for Moving Average prediction
evaluate_prediction(y_test, y_pred_ma)
```

```
{'root_mean_squared_error': 15.03, 'mean_absolute_percentage_error': 4.7}
```

As we can see above, both the LSTM and Moving Average prediction MAPE is less than 5%, which is good. Also the RMSE of LSTM is less than SMA model. We will do more comparison in the next section.

4.2 Justification

For this project, I have done two separate forecasting tests, one is for a 7 days forecast, and the other is a 15 days forecast. You can see the details in the notebooks.

The test performance of Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE) can be summarized as the table below.

	7 days forecast		15 days forecast	
	MAPE	RMSE	MAPE	RMSE
LSTM	3.9%	13.31	4.34	14.2
SMA	4.7%	15.03	7.04	19.39

As we can see from the above table, the LSTM model wins for both 7 days and 15 days predictions. In addition, when the prediction time is large, such as 15 days, the advantage of LSTM will be bigger.

Based on the above discussion, our LSTM based model for stock prediction is successful.

Part (V). Conclusion

I am glad to get the opportunity to build a stock predictor model through the Udacity nanodegree program. As we can see above, the model performance is ok and it successfully beat the benchmark model.

In addition to the model, I also built a web application, which uses the power of Dash and Plotly to enable users to run the model in an interactive way.

In the future, I plan to do more research for stock prediction, especially with machine learning and deep learning techniques, and hope I can apply it to my real investment.

Reference

[1]. https://www.investopedia.com/articles/basics/04/100804.asp
[2]. https://www.wsj.com/articles/few-sectors-have-escaped-augusts-stock-market-selloff-11567100241
[3]. https://github.com/plotly/dash
[4]. https://en.wikipedia.org/wiki/Root-mean-square_deviation
[5]. https://pypi.org/project/yfinance/
[6]. https://en.wikipedia.org/wiki/Long_short-term_memory
[7]. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[8]. https://github.com/plotly/dash
[9]. https://en.wikipedia.org/wiki/Moving_average