Capstone Project: Stock Price Prediction with Machine Learning

James Liang, Dec.2020

Section 1: Definition

1.1 Project Overview

As the advancement of software technologies and hardware capabilities brings along the era of Big Data, more and more practitioners devote themselves to the ocean of data, extracting information from data and ultimately distilling information into knowledge. It has come to our notice that quantitative modeling with machine learning algorithms on stock data has become more and more popular in the financial industry. In this project, we will use a set of methodologies backed with rigorous mathematical and financial theories to perform a forecast on stocks' close price traded in the U.S. market. We will use some widely recognized methods in feature engineering, together with state-of-art machine learning algorithms, to perform time-series forecasting on target stock assets.

1.2 Problem Statement

In this project, we will aim to predict the adjusted close price of stocks listed in the U.S. market. This project's objective can be summarized as a supervised learning problem where inputs are raw, historical data of a particular stock acquired through Yahoo! Finance API and output are numerical values of adjusted close price of the chosen stock in the next day. More specifically, the input data point will be a series that consists of several days of stock data (e.g., using data from 10 previous days to predict the adjusted close price of the next day). In this project, we restrain our stock choices to those in the U.S. market since the corresponding market suffers less from ambiguous regularizations, asymmetric information, and systematic risks comparing worldwide.

1.3 Metrics

To evaluate the performance of each of our prediction methods, we will use three evaluation functions in this project: Coefficient of Determination (R Squared) regression score, Mean Absolute Percentage Error (MAPE), and Mean Squared Error (MSE). The rationale of using three evaluation functions instead of one is to reduce bias in our evaluation metric: raw stock data series of different stocks varies significantly on price magnitudes and length. It is crucial to exclude exogenous factors such as differences in the size of datasets and variations of stock price magnitude. The desired result of our model should possess the following quantities:

1.High ***R Squared regression score***: The Coefficient of Determination (R Squared) score indicate the proportion of the variance in the dependent variable that is predictable from the independent variable

2.Relatively Low ***MSE***: Mean squared error place a more significant penalty on a large deviation between prediction and real value. Thus, a low MSE indicates the overall accuracy of the forecast and the rare occurrence of outliners in the forecast.

3. Low ***MAPE***: MAPE accounts of error in percentage, the fact that makes results of different stocks comparable (since it excludes the variation on price magnitude of different stocks)

Section 2: Analysis

2.1 Data Exploration

Below is an example of the Amazon stock dataset (ticker: AMZN). The original dataset was acquired through yahoo! Finance stock data API is a pandas data frame consist of columns. In this project, we use stock data sampled at a daily frequency. Furthermore, we will use columns in the raw dataset to compute some technical indicators as our features and properly cleanse, reformat our dataset after feature engineering. Below are explanations for each column in the raw data set:

1. ***Date***: Date information, as the timestamp for each row of stock data
2. ***Open***: The initial trading price of the underlying stock in a trading day
3. ***Close***: The last trading price of the underlying stock in a trading day
4. ***High***: The highest trading price of the underlying stock throughout a trading day
5. ***Low***: The lowest training price of the underlying stock throughout a trading day
6. ***Adj Close***: The last trading price of the underlying stock in a trading day, with the price value appropriately adjusted for corporate actions such as dividends, stock splits, and rights offering.
7. ***Volume***: The total number of shares traded through transactions of the underlying stock throughout a trading day

Though stock data acquired through well-known APIs are typically preprocessed with appropriate methods, designing a mechanism that deals with incomplete data set (i.e., NA and NaN values) are necessary. More importantly, certain feature engineering methods and outputs might also contain invalid or empty values by nature (i.e., features that demonstrate the average price level of a stock will have empty values in the first few data entries, where the requirement on time window length is not met). In this project, we adopt the general strategy of discarding data entries with NA or NaN values. Two critical reasons back the rationale behind this strategy: Firstly, given the relative abundance of data entries for most stock assets listed in the U.S market, it is acceptable to discard a small amount of data entries that are not complete. Secondly, time-series data records some fundamental and crucial aspects of real-world scenarios in its time order, thus makes them highly sensitive to data manipulation methods such as forward filling and backward filling. To avoid any possible data leakage, we choose to apply forward filling wherever possible and discarding incomplete data entries on the head and tail of data sets.

2.2 Exploratory Visualization

Raw stock data set used in this project are highly structured: If we define adjusted close price as our "original input," then other columns such as High and Low prices could be viewed as corresponding

"natural features" containing important information and implications the original input. To cater to the security market's traditions, we adapt a widely recognized form of visualization — Candlestick graph — as our primary visualization (see Figure 1). With methods of this form of visualization, all price information (Open, High, Low, Close) of one data entry are fused into one candle, filled with either green color (which implies an increase of close price compared to previous trading day) or red color (opposite as green). Such visualization allows people to access information from many different perspectives (e.g., overall trend, intraday oscillation, peak and bottom…)

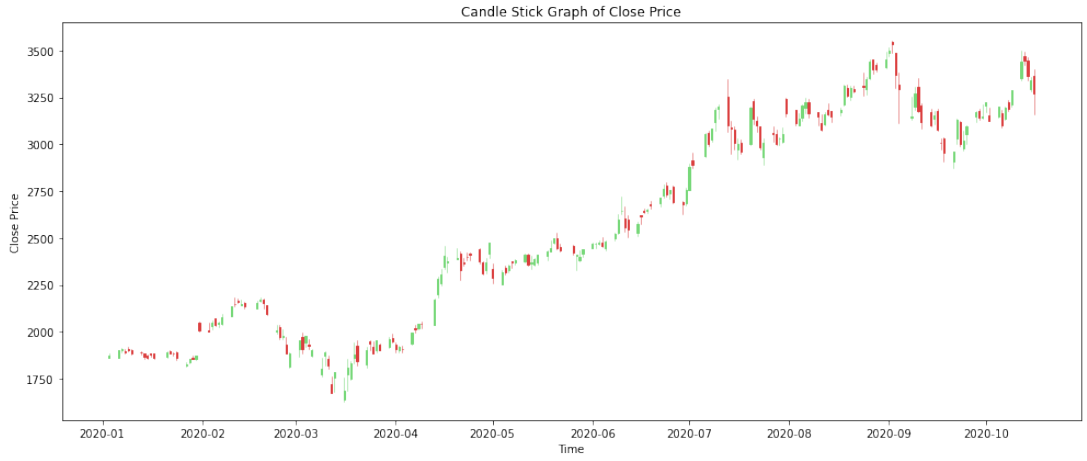


Figure 1: Candlestick Graph of Amazon Stock Close price, Jan 2020-Nov.2020

The second visualization (see Figure 2) is based on engineered features where three price average indexes are plotted over the same horizon. As the most important statistical feature and technical indicator of stock data, price average indexes are used by many different analysts for various purposes. It is worth mentioning that the two different prices average indexes: Simple Moving Average and Exponential Moving Average, have a slight difference in weight allocation rules.

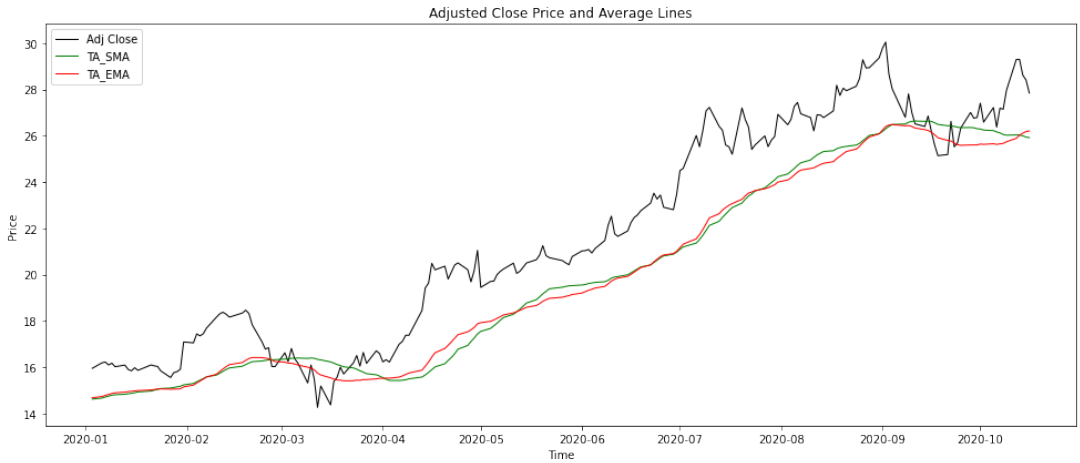


Figure 2: Close price and two types of price moving average of Amazon Stock Close price, Jan 2020-Nov.2020

The last visualization (see Figure 3) used more abstract features from our engineered features to visualize "trend": Since the stock market contains factors that are exogenous and hidden, such as public psychologies and informed traders[1], it is crucial to construct and visualize features that capture transaction trends for investigations and further inferences.

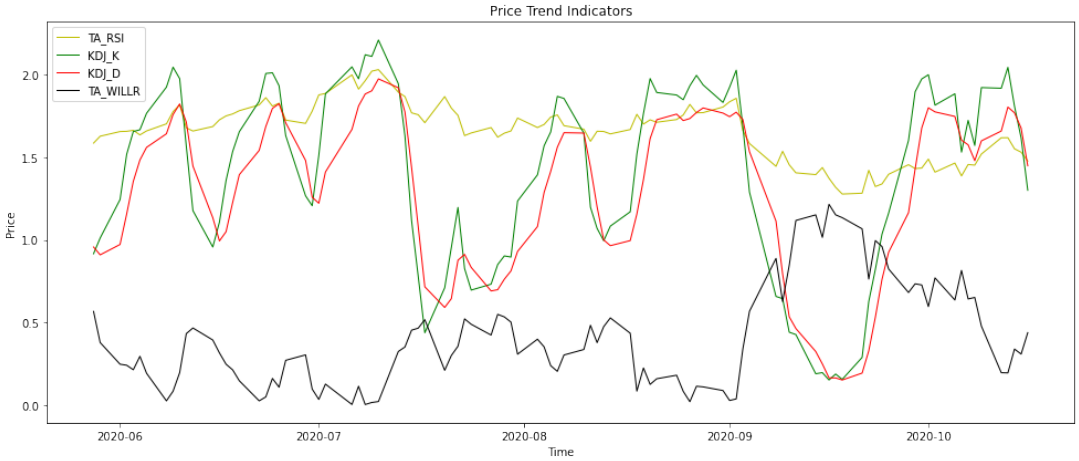


Figure 3: Various Price Trend indicators of Amazon Stock Close price, Jan 2020-Nov.2020

2.3 Algorithms and Techniques

In this project, we construct our primary forecast model as a Recurrent Neural Network (RNN), with Long Short Term Memory (LSTM) architecture framework. The primary reason for choosing an RNN model with LSTM is that it does not require many assumptions regarding autocorrelation and heteroscedasticity hold as autoregression does. Since the recurrent neural network can easily be overfitted, we shall set our default number of LSTM hidden layers to be three and the default number of nodes in each layer to be 50.

To train and backtest our stock data on our RNN model, we design functions that concatenate data rows and reshape the data frame on an appropriate length of "time window": The reshaped input data set has multiple day's featured stock data, with the last day as our prediction label and the rest of days as training input. Such a process of reshaping is demonstrated below (see Figure 4):

[1] Marcos Lo´pez De Prado, *Advances in Financial Machine Learning,* Page 29

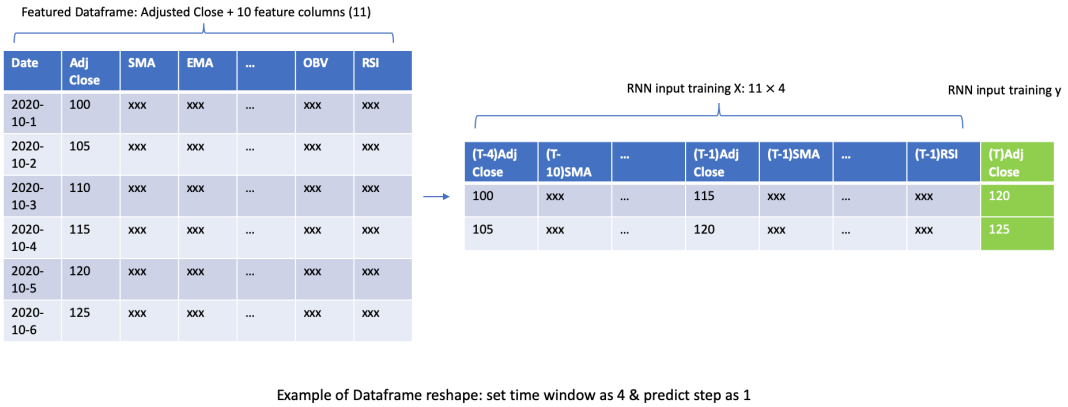


Figure 4: Data structure transformation process

2.4 Benchmark

Two extra models will be introduced to compare with our RNN with LSTM model: a non-algorithmic "model," where we use the Weighted Moving Average (WMA) of the adjusted close price within a specific time window as our forecasting results; an algorithmic model, where we use XGBoost regressor to perform stock price prediction, on the same set of data as our proposed RNN with LSTM model. To exclude exogenous factors in comparing two different algorithmic models' performance on the same stock, we will use the same featured dataset on both algorithmic models.

As a well-known machine learning model that has earned an excessive amount of reputation for its high accuracy, XGBoost regressor is introduced as a benchmark model to compare with our primary model on forecasting performance. It is worth mentioning that the other non-algorithmic "model" — WMA — is introduced not really as a forecasting model to compare with, however. Calculating historical price averages and using them as forecasting results does not involve a rigorous learning process but rather a process of guessing with agnostic believes upheld by many financial chartists. Given that security markets encompass a great number of latent factors (e.g., public psychology, information asymmetry, political influence…) that are incredibly intricate, it is worthwhile to introduce such a non-algorithmic model as a benchmark for reference.

Section 3: Methodology

3.1 Data Preprocessing

Various kinds of complicated preprocessing and feature engineering methodologies in academia and the financial industry are backed by mathematical theories such as orthogonal transformation and Hilbert transformation. In this project, we will adopt a relatively simple and straightforward feature engineering strategy. We will calculate a set of carefully selected technical indicators of stock data used in stocks' fundamental analysis. The rationale for using relatively simple preprocessing methodologies is that advanced technologies such as wavelet transformation have not been proved robust against data leakage problems[2]

Our technical indicators choices come in four groups: price-average indicators, transaction behavior & trend indicators, volume indicators, and Hilbert transform indicators. Below are explanations of feature groups:

1. **Price average indicator(s)**: Indicate the average historical performance of a stock within a given time window. Price average allows the model to capture information regarding the growth speed of stocks within a market circumstance (i.e., in most cases, we saw price average goes up overtime except in financial crisis, which reveals the fact that stock assets grow in value as the economy grows at a stable rate)
2. **Transaction behavior & trend indicator(s)**: Reveal some information regarding market participants' behaviors, along with the price status of stocks resulting from those behaviors.
3. **Volume Indicator**: Reveals essential information regarding the market conditions (the volume can be used to infer important information on the industrial sector/trading market / geological region of the underlying stock. That information can reveal systematic factors rather than idiosyncratic ones.
4. **Hilbert Transformation indicator(s)**: A set of technical indicators backed by exclusive mathematical theories on signal processing, where the stock price is viewed as a sequence that possesses hidden periodicity

Additionally, we introduced a unit scaler where we divide all rows of data by its value in the first row (Except feature columns of Hilbert Transformation indicators). Such a scaler makes visualization more readable, and the optimization process executes faster on our algorithmic models. Notice that besides this unit scaler, we introduce a min-max scaler from the Scikit-Learn package to perform an extra step of value normalization on the dataset prepared for our RNN model to boost training speed further.

---

[2] Ryo Hasumi and Yuto Kajita, *Boundary problem and data leakage: A caveat for wavelet-based forecasting,* Section 3: Wavelet-based forecasting and data leakage: A simple example

3.2 Implementation

Implementation of primary and benchmark model consists of four steps in general, and here we will use the implementation of our primary model as an example.

Step 1. General and customized data processing: Split the featured data set acquired through predefined feature engineering methods, perform min-max scaling, and finally perform "stacking" (to transform the forecasting problem into a supervised learning problem) on training and testing datasets, respectively. (see Figure 5)

```
# Prepare dataset for RNN model
AMZN_rnn_df = AMZN_features_data.loc[:,'Adj Close':'TA_HT_TRENDMODE']

# Set learning window length as 10 (trading days)
window_len = 10
n_features = AMZN_rnn_df.shape[1]
rnn_split_ratio = 0.75

# step 1: perform train-test split
raw_rnn_train_set, raw_rnn_test_set = train_test_split (AMZN_rnn_df, split_ratio = rnn_split_ratio)

# step 2: feature scaling, using min-max scaler
train_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
raw_rnn_train_set = train_scaler.fit_transform(raw_rnn_train_set)

test_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
raw_rnn_test_set = test_scaler.fit_transform(raw_rnn_test_set)

# step 3: perform "stacking" (transform a forecasting problem into a supervised problem)
rnn_train_set = train_test_prep(raw_rnn_train_set,
                                window_len = window_len,
                                output_step = 1)

rnn_test_set = train_test_prep(raw_rnn_test_set,
                                   window_len = window_len,
                                   output_step = 1)
```

```
# spliting inputs from outputs
rnn_train_X, rnn_test_X, rnn_train_y, rnn_test_y = train_test_reformat(rnn_train_set.values,
                                                                       rnn_test_set.values,
                                                                       window_len = 10,
                                                                       output_step = 1)

print(rnn_test_X.shape)
```

Figure 5: Data split and normalization process

Step 2. Define the model and training process: Define an RNN model with LSTM hidden layers and enable dropout to avoid gradient problems (vanishing & exploding). In this application we will use Adam optimizer for training since it possesses benefits of the gradient momentum and the RMS prop optimization technique. After defining the model, we shall define a training process with appropriate hyper-parameters. (see Figure 6)

```
model = keras.models.Sequential()
model.add(keras.layers.LSTM(units = 50,
                            return_sequences = True,
                            input_shape=(rnn_train_X.shape[1],rnn_train_X.shape[2])))
model.add(keras.layers.Dropout(0.2))

model.add(keras.layers.LSTM(units = 50))
model.add(keras.layers.Dropout(0.2))

model.add(keras.layers.Dense(1))
model.compile(loss='mse', optimizer='adam')
```

```
# fit network, notice: epochs should not be set to large
rnn_train_history = model.fit(rnn_train_X,
                              rnn_train_y,
                              epochs=10,
                              batch_size=60,
                              validation_data=(rnn_test_X, rnn_test_y),
                              verbose=2,
                              shuffle=False)
```

Figure 6: Model construction and Hyperparameters initialization

It is worthwhile to plot training and validation losses along training epochs to monitor the training process. Ideally, the error curve should be sloped downward until it gets nearly flat. (see Figure 7)

```
plt.plot(rnn_train_history.history['loss'], label='train loss')
plt.plot(rnn_train_history.history['val_loss'], label='validation loss')
plt.legend()
plt.show()
```

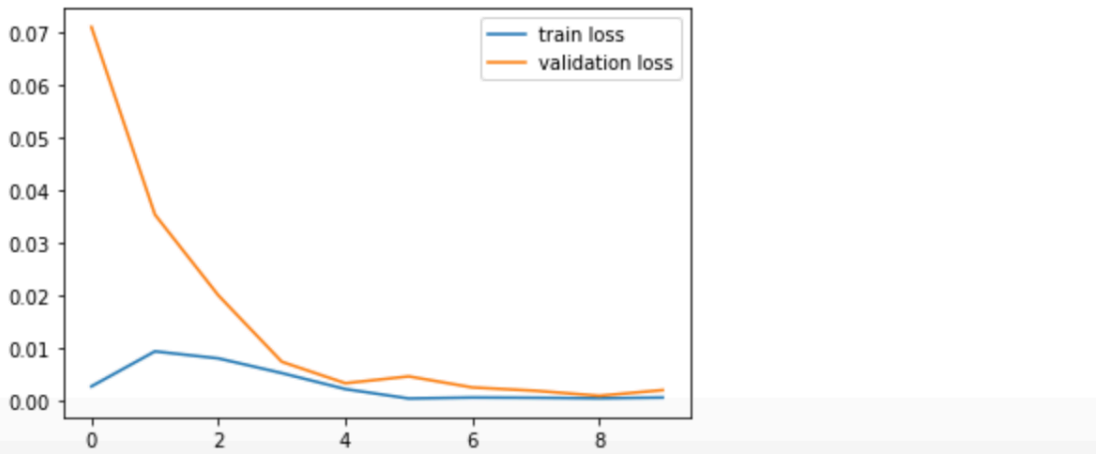


Figure 7: training and validation loss curves

Step 3. Perform prediction on test dataset: Perform prediction on test dataset with the trained model, reshape outputs into proper shape for the evaluation process. (see Figure 8)

```
# use inputs of test set to perform prediction
rnn_y_predict = model.predict(rnn_test_X)
rnn_test_X = rnn_test_X.reshape((rnn_test_X.shape[0], window_len*n_features))

rnn_test_y = rnn_test_y.reshape((len(rnn_test_y), 1))
```

Figure 8: Applied trained model for prediction and reshape output

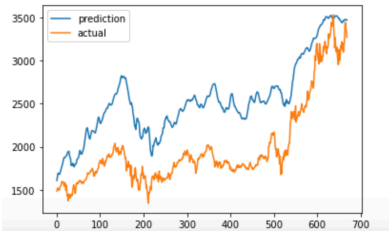
Step 4. Evaluate performance using evaluation functions: Use prediction output values and actual output values to calculate evaluation functions. Notice that any customized processing steps conducted earlier in the implementation stage need to be reversed. In our case, we need to inverse scaling both prediction and actual output values. (see Figure 9)

```
# invert scaling for forecast
rnn_inv_y_predict = np.concatenate((rnn_y_predict, rnn_test_X[:, -14:]), axis=1)
rnn_inv_y_predict = scaler.inverse_transform(rnn_inv_y_predict)
rnn_inv_y_predict = rnn_inv_y_predict[:,0] * AMZN_base_value_dict['Adj Close']

# invert scaling for actual
rnn_inv_y = np.concatenate((rnn_test_y, rnn_test_X[:, -14:]), axis=1)
rnn_inv_y = scaler.inverse_transform(rnn_inv_y)
rnn_inv_y = rnn_inv_y[:,0] * AMZN_base_value_dict['Adj Close']

# calculate evaluation functions
rnn_mse_error = mean_squared_error(rnn_inv_y, rnn_inv_y_predict)
rnn_mape_error = mean_absolute_percentage_error(rnn_inv_y, rnn_inv_y_predict)
rnn_r_squared_score = r2_score(rnn_inv_y, rnn_inv_y_predict)

rnn_error_dict = {'rnn_mse_error':rnn_mse_error, 'rnn_mape_error':rnn_mape_error, 'rnn_r_squared_score':rnn_r_squared_score}
print('Test Mean_Squared_Error: %.3f' % rnn_mse_error)
print('Test Mean_Absolute_Percentage_Error: %.3f' % rnn_mape_error)
print('Test R_Squared regression score: %.3f' % rnn_r_squared_score)
```

Evaluation Result & Visualization

```
Test Mean_Squared_Error: 342999.514
Test Mean_Absolute_Percentage_Error: 0.298
Test R_Squared regression score: -0.425
```

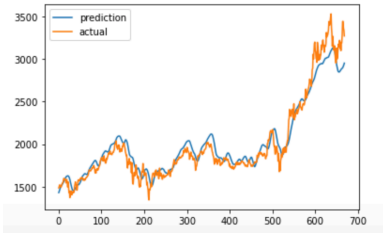
```
plt.plot(rnn_inv_y_predict, label='prediction')
plt.plot(rnn_inv_y, label='actual')
plt.legend()
plt.show()
```



Number of hidden LSTM layers: 4
Number of epochs: 20

```
Test Mean_Squared_Error: 15700.841
Test Mean_Absolute_Percentage_Error: 0.045
Test R_Squared regression score: 0.935
```

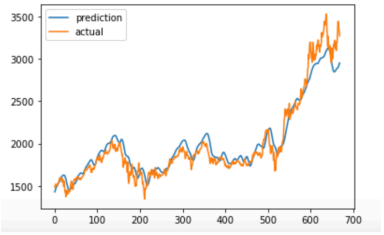
```
plt.plot(rnn_inv_y_predict, label='prediction')
plt.plot(rnn_inv_y, label='actual')
plt.legend()
plt.show()
```



Number of hidden LSTM layers: 3
Number of epochs: 20

```
Test Mean_Squared_Error: 9771.293
Test Mean_Absolute_Percentage_Error: 0.034
Test R_Squared regression score: 0.959
```

```
plt.plot(rnn_inv_y_predict, label='prediction')
plt.plot(rnn_inv_y, label='actual')
plt.legend()
plt.show()
```



Number of hidden LSTM layers: 2
Number of epochs: 20

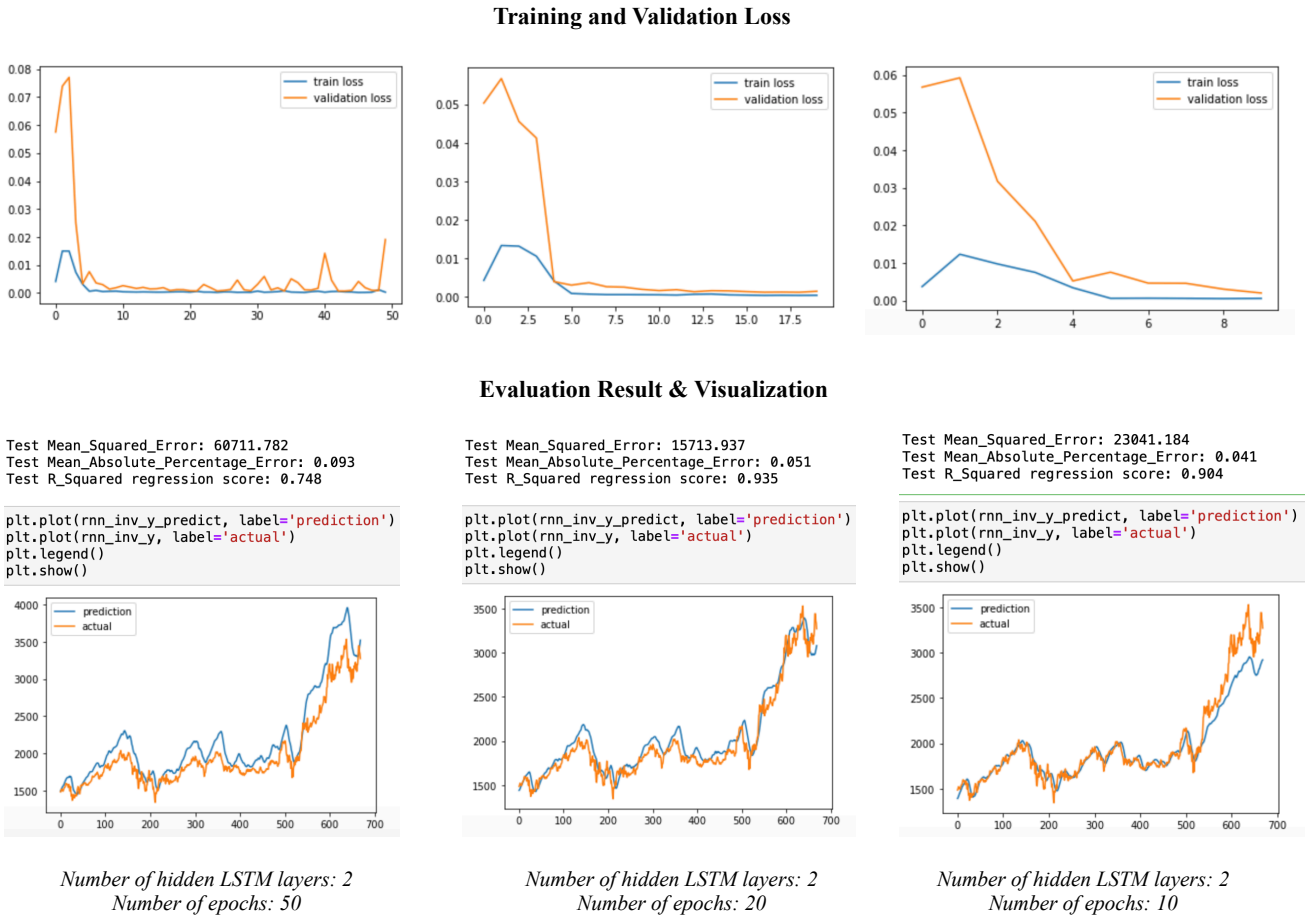
Figure 9: Apply evaluation metric and visualize result

3.3 Refinement

Throughout the implementation stage, two crucial hyper-parameters of the primary model has been refined through feedback from evaluation metric (see Figure 10):

1. **The number of hidden LSTM layers:** Many studies have shown that recurrent neural networks can easily overfit the dataset if too many hidden layers are built into the network. The number of hidden layers is eventually set as two after many independent trials of the train-test experiment in the implementation stage. We started with four hidden layers and decreased one layer at each trial. The evaluation results and visualization is shown below.

2. **Number of training epochs**: With many independent trials of train-test experiment throughout the implementation stage, it was found that inappropriate numbers of training epochs could easily lead to poor prediction performance, either due to over-fitting or under-fitting (a high validation error typically demonstrates over-fitting but a low training error; under-fitting is typically demonstrated by a high validation and high training error)



Number of hidden LSTM layers: 2
Number of epochs: 50

Number of hidden LSTM layers: 2
Number of epochs: 20

Number of hidden LSTM layers: 2
Number of epochs: 10

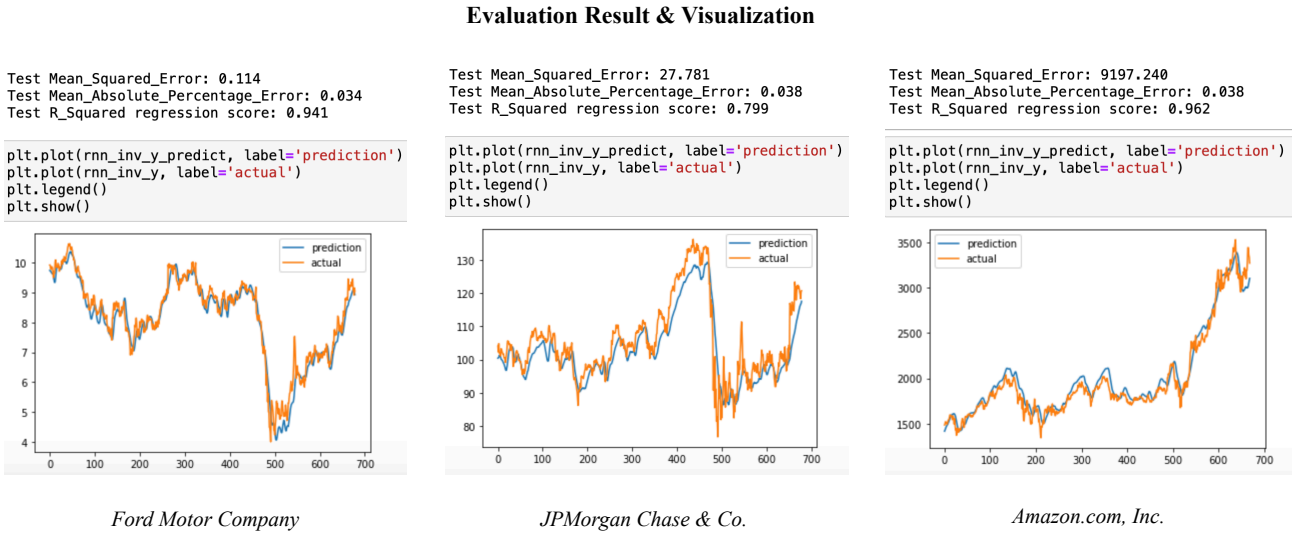
Figure 10: Hyperparameter tuning and result visualizations

Section 4: Results

4.1 Model Evaluation and Validation

After carefully review information acquired from abundant train-test experimental trials conducted through the implementation and refinement stage, we eventually set the number of hidden LSTM layers to be two and the number of training epochs to be 15. The RNN model is then simulated on three different stock assets with a great variation on industrial&business coverage, price growth trend, and price magnitude:

1. **Amazon.com, INC**: Technology industry / E-commerce & Cloud Service / High stock price value / fast growth trend
2. **JPMorgan Chase & Co**: Financial Service industry / Commercial & Investment bank / medium stock price value / slow growth trend
3. **Ford Motor Company**: Manufacturing industry / Automobile Manufacturer / low stock price value / flat trend



Ford Motor Company *JPMorgan Chase & Co.* *Amazon.com, Inc.*

Figure 11: Model Assessment with visualizations

According to results from our evaluation metric, which consist of three evaluation functions, The overall performance of our RNN model is satisfactory in general:

1. Mean squared errors have fell into acceptable value ranges accordingly (Ford < 1 ; JPMorgan < 100 ; Amazon < 10000)
2. Mean absolute percentage errors are low (< 5 %)
3. Feature explainability is high (> 75%)

Thus, the model is proven to be robust and well-generalized

4.2 Justification

In this project, we introduce two benchmark models: a non-algorithmic "model" that uses the Weighted Moving Average (WMA) indicator as predictions of future prices and an algorithmic model based on XGBoost regressor. To evaluate our primary model's performance, we use the Amazon stock dataset to perform a train & test experiment on all three models. The result is indicated below.

	mean squared error	mean absolute percentage error	r2 score
RNN	9093.240080	0.036755	0.970160
WMA	10539.075718	0.033934	0.969481
XGB	3440.967248	0.018862	0.990000

Figure 12: Model comparisons (Primary model v.s benchmark models) on evaluation metrics

Notice that of all three models, the WMA “model” achieves a better performance than the RNN model under our current evaluation metric. As we have stated above, the WMA “model” does not perform rigorous learning on historical data. However, the evaluation result implies that a pure machine learning model might not always outperform a simple statistical model for stock assets that possess specific latent properties. Once again, this implication reminded us that stock price prediction is an immense task. Nevertheless, no entities have developed machine learning algorithms that are accurate and reliable on every stock asset.

Section 5: Conclusion

5.1 Free-Form Visualization

Below we plot prediction results of the primary RNN model and two benchmark models of Amazon stock data correspondingly. Notice that the y-axis in all graphs represents stock price, and the x-axis is the index of prediction results. Overall, all three models performed well on the same set of stock data.

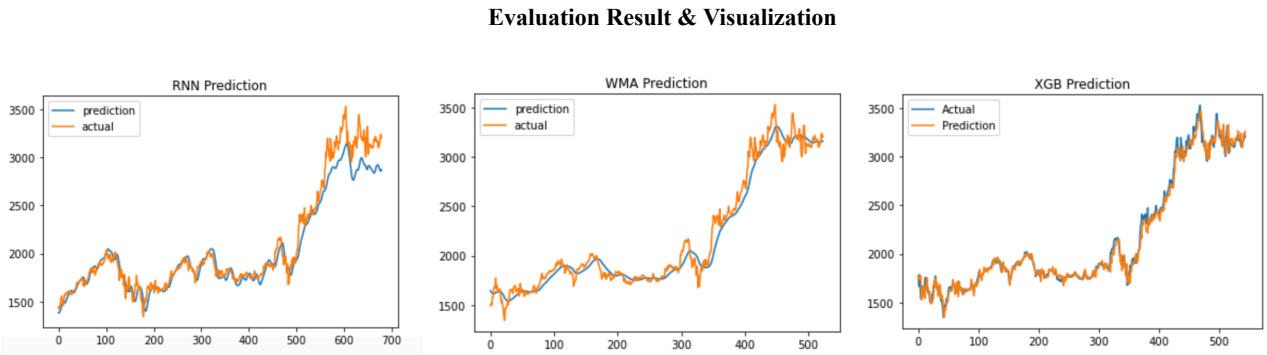


Figure 13: Model prediction (Primary model v.s benchmark models) on Amazon Stock data

5.2 Reflection

Stock price prediction is an immense task that is far more difficult than most people would initially expect: the stock market consists of many complicated yet latent factors that influence the prices of assets. Throughout this project, I have utilized many skills and acquired knowledge to design and construct stock price predictors in an exploratory research setting: First, by defining data extraction, preprocessing, and visualization methods, I managed to complete exploratory data analysis and acquired enlightenments on feature engineering methods and machine learning models to be used. Second, by constructing the primary model, defining evaluation metrics, and tuning the model hyper-parameters based on many independent trials of train-test experiments, I finished constructing my primary prediction model and received satisfactory prediction performance. Lastly, by carefully selected two benchmark models, deployed them, and have all models tested on multiple stock samples that vary significantly, I finally get to exclude my doubts about the generality and validate the primary model's robustness.

One major challenge I encountered was algorithm choice, which had a fundamental impact on the architecture and performance of the framework. Specifically, traditional solutions for such tasks often involve some variant of autoregressive regression, which is essentially an application of linear

regression. As a result, it has very stringent assumptions on the input data, such as autocorrelation and heteroscedasticity. Though this approach is widely applied in the industry and even Basel III, I was concerned about the implication of these assumptions because it appears to be an overly optimistic assertion of the input data. As a result, I chose to adopt an RNN model Long Short-Term Memory (LSTM), because it did not require that many assumptions hold as autoregression does and would save a great effort for feature engineering.

5.3 Improvement

Two improvements could further boost the primary prediction model's performance in this project: Firstly, it would further improve the primary prediction model's robustness by introducing an extra step of hyperparameter grid-search. It has come to my attention that many industrial practices have adopted such a strategy when deploying their machine learning models for specific tasks in OLAP systems that do not requires millisecond responses. Though it requires extra setups on parameters and a more complicated evaluation metric, it has a great potential of optimizing the proposed RNN model in this project.

Secondly, it is possible to improve the RNN model by introducing artificial features extracted from text data of news. My current working responsibilities in the real world are associated with some tasks involving natural language processing. This fact allows me to get to know some state-of-art industrial applications of public sentiment analysis using news data to perform sales forecasting and operation simulation modeling. It would be highly complicated to integrate stock data that are highly structured with text data that are mostly unstructured into one framework, but doing so would probably improve our prediction model's performance by a surprising amount.

5.4 Error Correction

In the development stage of this project, one hidden but rather critical problem was found and later corrected in the implementation stage of the RNN model. Originally, the data preparation procedures were designed as the following:

1. Perform “Stacking” on the featured dataset
2. Perform min-max scaling on a featured dataset
3. Split the whole dataset into a training set and a testing set, and then split inputs from outputs, respectively

```
# ensure all data is float
#values = values.astype('float32')

AMZN_rnn_df = AMZN_features_data.loc[:,'Adj Close':'TA_HT_TRENDMODE']

# Set learning window length as 10 (trading days)
window_len = 10
n_features = AMZN_rnn_df.shape[1]

# feature scaling, using min-max scaler
scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
AMZN_rnn_df = scaler.fit_transform(AMZN_rnn_df )

# frame as supervised learning
rnn_train_X, rnn_test_X, rnn_train_y, rnn_test_y = train_test_prep(AMZN_rnn_df,
                                                                   algo_name = 'lstm',
                                                                   window_len = window_len,
                                                                   output_step = 1,
                                                                   split_ratio = 0.75)

print(rnn_train_X.shape)
```

Figure 13: Wrong implementation of data preparation steps (cause data leakage)

Notice that the train-test split of the dataset happened AFTER the min-max scaling. Such a procedure will cause data leakage, where information regarding the testing set gets passed into the training set. A logical illustration is provided below:

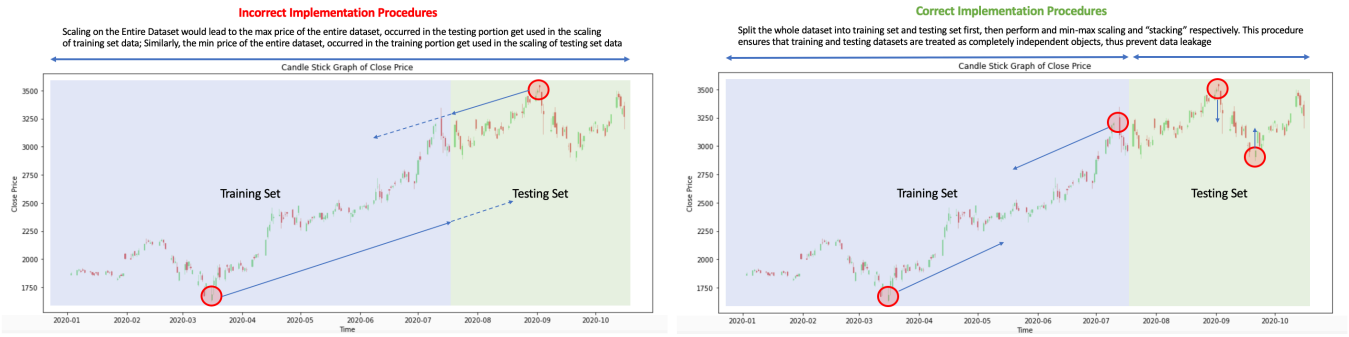


Figure 14: Logic clarification of data leakage problem

Thus, the correct implementation procedures are:

1. Split the whole featured dataset into a training set and a testing set, and then split inputs from outputs, respectively
2. Perform min-max scaling on training featured dataset and testing featured dataset, correspondingly
3. Perform “Stacking” on the featured dataset

Reference

1. Marcos Lo´pez De Prado*, Advances in Financial Machine Learning,* Page 29
2. Ryo Hasumi and Yuto Kajita, *Boundary problem and data leakage: A caveat for wavelet-based forecasting*, Section 3: Wavelet-based forecasting and data leakage: A simple example