STOCK MARKET PREDICTION USING LSTM

Harsh Sharma1

1 Computer Science Department, Purdue University Fort Wayne

Fort Wayne, IN, 46805,

sharh02@pfw.edu

**Abstract.** This data science project employs Long Short-Term Memory (LSTM) Recurrent Neural Networks to develop an advanced stock market prediction model. Utilizing data sourced from the New York Stock Exchange (NYSE), the model aims to capture intricate temporal patterns within stock price movements. By leveraging LSTM's ability to retain past information and recognize sequential dependencies, the project seeks to enhance forecasting accuracy. The methodology involves preprocessing the NYSE data, constructing and training the LSTM RNN architecture, and rigorously evaluating its predictive performance using RMSE. Successful implementation could offer valuable insights for investors and financial analysts, potentially improving decision-making strategies in the dynamic realm of stock market trading.

**Keywords: LSTM, RMSE, NYSE, RNN**

1 Introduction

* Project problem description

The stock market is known for its complex and volatile nature, where prices are influenced by a multitude of factors including economic indicators, geopolitical events, and investor sentiment. Traditional models often struggle to capture the intricate temporal patterns and non-linear dependencies present in stock price data.

* Motivation

The motivation behind this project stems from the critical importance of accurate stock market predictions for investors, financial institutions, and the broader economy. The stock market's unpredictable nature and potential for significant financial gains or losses make reliable forecasting a highly incentivized endeavor. Additionally, the project's findings could contribute to the broader field of financial forecasting and machine learning, advancing our understanding of how sophisticated models can enhance predictions in complex, real-world scenarios.

* Application Domain

The application domain of this project lies at the intersection of finance, data science, and machine learning. It primarily targets the field of quantitative finance, stock trading, and algorithmic trading. The project's outcomes and the developed LSTM model have potential applications in various areas: Financial Technology(FinTech), I**nvestment Strategies, Academic Research.**

* Contributions of the work to Computer Science domain

The project could contribute to academic research by exploring the effectiveness of LSTM RNNs in stock market prediction and advancing the understanding of financial time series forecasting. Also, it helps to showcase the the collaboration between Computer Science and Finance, demonstrating the broader impact of computational techniques in real-world decision-making.

* Data Description

The data used for this project is open-source and is readily available by **Yahoo Finance**. Although, the source of the data is **New York Stock Exchange** (NYSE). NYSE generates and records the data in real-time. And Yahoo Finance is the third party which helps users to access the data easily. To import the data directly from Yahoo Finance I’m using a python library called ‘**yfinance’**.

* Short Description of my Approach

After fetching the data using yfinance, I’m applying MinMax Normalization as a preprocessing step. Also, as a data exploration step I’m visualizing the time-series data of closing price of APPLE stock. And then training the LSTM after defining it’s architecture. I’m using Root Mean Square Error (**RMSE**) as a performance metric.

* Short Description of my findings

I had tune in a couple of hyper parameters to get a better performing LSTM trained model. I also expermineted with amount of data which was used to train the model. And I found the optimal data was from 2010 to current date. If I used more data, the performance of the model decreased. It could be because the old data was obsolete and wasn’t of much useful in predicting the current stock market price.

1. Problem Statement

The stock market is very volatile and depends on a lot of factors like socio-economic state of a country, the current policy by government, trader’s sentiment etc. And so, it’s very difficult to capture the dynamics of stock market and to be able to predict it at the very least.

The primary goal is to design and optimize an LSTM architecture that can reliably forecast future stock prices based on past NYSE data. This entails overcoming the inherent volatility and non-linearity of the stock market, ultimately providing a valuable tool for investors, traders, and financial analysts to make informed decisions in a dynamic and competitive market environment.

Objectives:

 **Data Collection and Preprocessing**: Gather historical stock price data from the NYSE and preprocess it to ensure accuracy, consistency, and suitability for training the LSTM model.

 **Model Architecture Design**: Develop a sophisticated LSTM RNN architecture that can effectively capture intricate temporal patterns and dependencies within the stock price data. I’m using 50 neurons for the LSTM architecture.

 **Hyperparameter Tuning**: Optimize the model's hyperparameters to enhance its predictive performance and generalization capabilities.

 **Training and Validation**: Train the LSTM model on the preprocessed data, utilizing techniques such as sequence padding, batching, and cross-validation to ensure robust training and validation.

 **Performance Evaluation**: Rigorously evaluate the model's predictive accuracy and effectiveness using appropriate metrics, considering factors like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

**Given,**

In here I have the NYSE data of APPLE Inc.’s stock data. The closing price is of interest, and the rest of data for the stock is discarded.

**Goal,**

The goal is to successfully predict the closing price of APPLE Inc.

**Assumption,**

I’m only considering one input feature which is the closing price. The assumption is that the closing prices should be enough for predicting the stock price precisely.

1. Related Work

A couple of work which I referred and are also related work are listed below:

 **Stock Price Prediction Using LSTM, RNN and CNN-Sliding Window Model"**: This paper explores stock price prediction using LSTM, RNN, and CNN-based sliding window models. It compares the performance of these models and investigates their ability to capture temporal patterns in stock price data.

 **"A Deep Learning Framework for Financial Time Series using Stacked Autoencoders and Long Short-Term Memory"**: The paper introduces a deep learning framework combining stacked autoencoders and LSTM networks for financial time-series prediction. It focuses on feature representation and temporal dependencies to enhance prediction accuracy.

 **"Stock Price Prediction Using Attention-based Multi-Input LSTM"**: This research proposes an attention-based multi-input LSTM architecture for stock price prediction. It considers both stock prices and external factors like news sentiment to improve the model's predictive capabilities.

 **"Time Series Prediction Using LSTM Networks: Application to Stock Market"**: The paper investigates the application of LSTM networks to stock market prediction. It emphasizes data preprocessing, model training, and evaluation, highlighting the challenges and opportunities in this context.

 **"Deep Learning for Stock Market Prediction Using Technical Indicators and Financial News Articles"**: This study combines deep learning techniques with technical indicators and financial news articles for stock market prediction. It explores the integration of multiple data sources to enhance forecasting accuracy.

 **"Predicting Stock Prices Using a Feature Fusion LSTM-CNN Model"**: The paper presents a model that fuses LSTM and CNN architectures for stock price prediction. It discusses the benefits of leveraging spatial and temporal patterns in financial data.

1. Background Concepts (LSTM Architecture)

I needed to understand the architecture of LSTM in order to apply it efficiently. LSTM is a recurrent neural network (RNN) architecture designed to handle long-term dependencies in sequential data. And it’s good, since the stock price data is also a sequential data. LSTM is also good for processing text, voice, video or any other sequential data. It’s also used in text generation. Here’s the architecture discussed in terms of text prediction.

Consider the below to be the dataset:

Input Sequence: "I like"

Output Sequence: "AI.”

Input Sequence: "Machine learning"

Output Sequence: "is amazing.”

Input Sequence: "Recurrent neural networks"

Output Sequence: "can process sequential data."

STEP 1

We tokenize the data and perform indexing.

"I": 0, “like": 1, "Machine": 2, "learning": 3, "Recurrent": 4, "neural": 5, "networks": 6, "AI.": 7, "is": 8, "amazing.": 9, "can": 10, "process": 11, "sequential": 12, "data.": 13

In this lexicon, only the unique tokens are stored and all the repeated iterations are truncated.

STEP 2

After this from embeddings and form a dense vector for each input and output sequence:

"I like" becomes [0, 1] and "Machine learning" as [2, 3] and “AI” as [7]

STEP 3

Then we feed the model these embeddings. The architecture of LSTM consists of three grates. They are: Input, Forget and Output gate.

* Forget Gate: The forget gate determines what information from the previous cell state should be discarded or forgotten. It takes the previous cell state (C(t-1)) and the current input (X(t)) and passes them through a sigmoid activation function.
* Input Gate: The input gate determines what new information should be stored in the cell state. It consists of two parts: the input modulation and the new candidate values. The input modulation involves passing the previous cell state (C(t-1)) and the current input (X(t)) through a sigmoid activation function.
* Output Gate: The output gate determines what information from the current cell state should be used as the output of the LSTM cell. It takes the current input (X(t)) and the previous cell state (C(t-1)), and also considers the updated cell state (C(t)) after applying the forget and input gates.

STEP 4

Generate Probabilistic Distribution using the above lexicons. For example: If we are trying to predict the next word after "I like," the model might produce a probability distribution like [0.05, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.85, 0.01, 0.03, 0.01, 0.01, 0.01].

The probability distribution of AI is obviously high since it has already been present in the dataset. The word with highest probability distribution is then selected or predicted.

In this project, I’m taking the input sequence of 60 days, and each data point in the sequence is the closing price of Apple stock.

1. Data

**5.1 Data**

* + The data used for this project is open-source and is readily available by **Yahoo Finance**. Although, the source of the data is **New York Stock Exchange** (NYSE). NYSE generates and records the data in real-time. And Yahoo Finance is the third party which helps users to access the data easily. To import the data directly from Yahoo Finance I’m using a python library called ‘**yfinance’**.
  + The Yfinance library conveniently downloads the data in python dataframes making it easier to operate on it.
  + The dataset considered for the project is from date range: 2010-01-01 to 2023-07-15. It has a total of 3405 rows. The dataset has 7 attributes and they are: Date, Open, High, Low, Close, Adj Close, Volume.
  + Note that the Date is also the index of the dataset.
  + For training the model, 80% of dataset is considered. Meanwhile rest 20% of data is used to evaluate the trained model.
  + Here are the top 5 data entries as sample data:

|  | **Open** | **High** | **Low** | **Close** | **Adj Close** | **Volume** |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** |  |  |  |  |  |  |
| **2010-01-04** | 7.622500 | 7.660714 | 7.585000 | 7.643214 | 6.496295 | 493729600 |
| **2010-01-05** | 7.664286 | 7.699643 | 7.616071 | 7.656429 | 6.507525 | 601904800 |
| **2010-01-06** | 7.656429 | 7.686786 | 7.526786 | 7.534643 | 6.404016 | 552160000 |
| **2010-01-07** | 7.562500 | 7.571429 | 7.466071 | 7.520714 | 6.392175 | 477131200 |
| **2010-01-08** | 7.510714 | 7.571429 | 7.466429 | 7.570714 | 6.434673 | 447610800 |

**5.2 Data Exploration**

* As for data exploration, I tried to visualize the data using time-series plot. Where I plotted closing prices of Apple stock against the date.

A graph showing a line

Description automatically generated

1. Data preprocess

* I’ve used MinMax Normalization to preprocess the data. It helped to bring the scale of dataset in the range of 0 and 1. This normalization removed the magnitudes of dataset which is better for training the model.
* Here is the result of data as how it was changed before and after normalization.

A screenshot of a computer screen

Description automatically generated A screenshot of a computer code

Description automatically generated

(Before) (After)

* Also, for training the model 6 features would been excess and would have created problem in finding relations with stock price. The model would have performed poorly with so many features. And so, I used only one feature which is ‘Close’ price. I had to truncate the rest of the features including: volume, open, adj close etc. This is the result of working data:

A screenshot of a computer

Description automatically generated

1. Methodology
   * To train the model I selected 60 days of input sequence. If I selected less window size then the model wouldn’t have enough data to find key points. And if I selected a larger window size then I would have risked **overfitting** the model.
   * Here is the pseudo code for preparing the data for training:

train\_data = scaled\_data[0:training\_data\_len, :]

x\_train = []

y\_train = []

for i in range(60, len(train\_data)):

x\_train.append(train\_data[i-60:i, 0])

y\_train.append(train\_data[i, 0])

Here, x\_train contains the input features of past data whereas y\_train contains the corresponding target values.

* + Then, I also had to create the architecture for the LSTM model. I used 50 neurons for training the model. Here is the pseudo code of the same:

# Create the model

model = Sequential()

# Add the first LSTM layer

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(x\_train\_shape\_1, 1)))

# Add the second LSTM layer

model.add(LSTM(units=50, return\_sequences=False))

# Add a Dense layer with 25 units

model.add(Dense(units=25))

# Add the output Dense layer

model.add(Dense(units=1))

* + Also, I had normalized the data to train the model, in order to evaluate the model I needed to get back the data in its original form. That means I had to inverse the MinMax normalization.
  + In order to evaluate the performance of newly trained model I used Root Mean Square error. Here is the pseudo code which I used to calculate the rmse:

rmse = np.sqrt(np.mean((predictions - y\_test)\*\*2))

1. Experiment

In order to improve the performance of the trained model I had to tune in a couple of hyperparameters and reconsider the amount of data which I was using to train the model. I used RMSE to compare the performance of the various iterations of the model.

* 1. **Experiment Goals**

The goal of the experiment was to improve the accuracy of the model. In the first two experiments I just changed the number of epochs to see how the performance of model will vary. Then in the next two experiments I changed the date range, essentially changing the amount of the data which is used to train the model. Here are the various iterations:

* + 1. **Experiment #1**

Number of epochs: 10

Date Range: 2012-01-01 to 2023-07-15

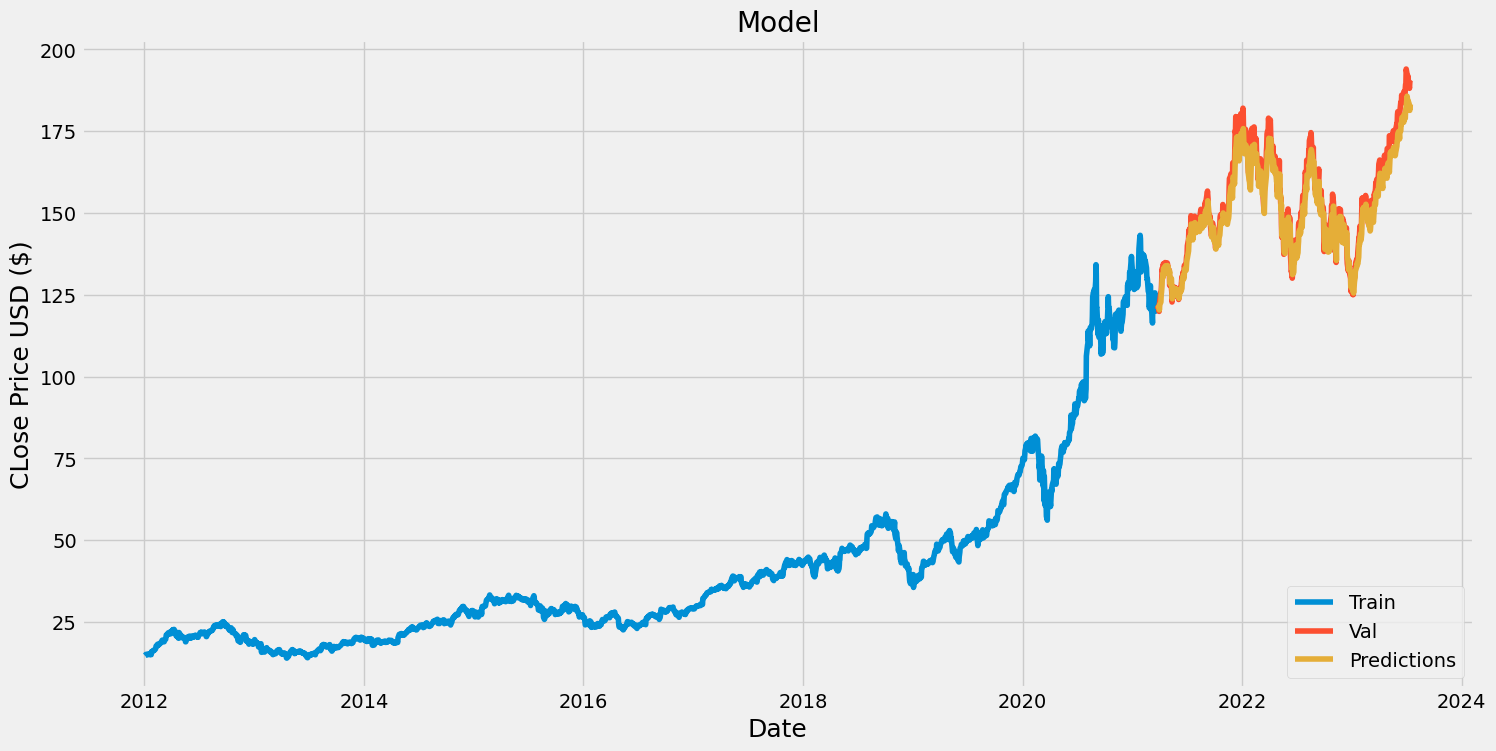
RMSE: 5.26

* + 1. **Experiment #2**

Number of epochs: 2

Date Range: 2012-01-01 to 2023-07-15

RMSE: 2.78

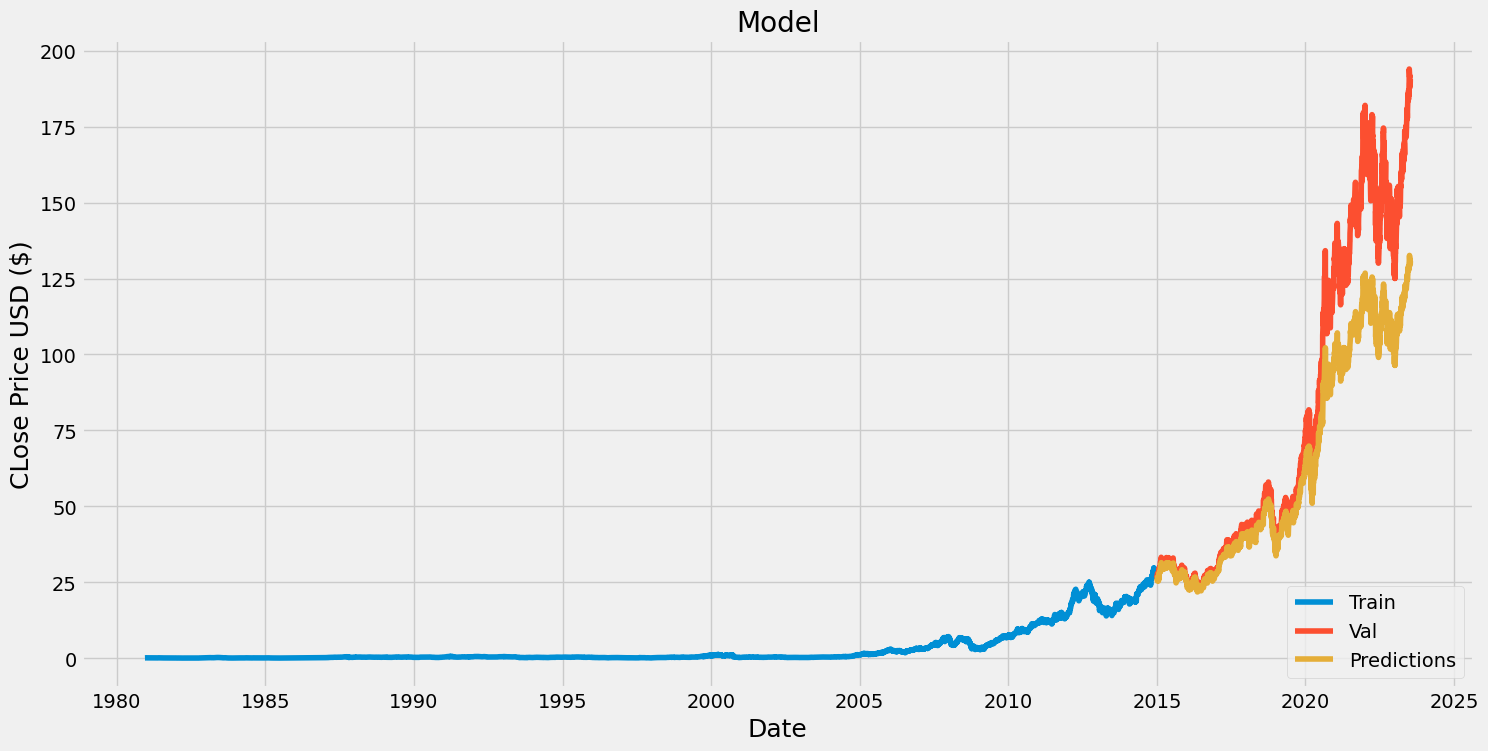


* + 1. **Experiment #3**

Number of epochs: 2

Date Range: 1980-01-01 to 2023-07-15

RMSE: 15.41

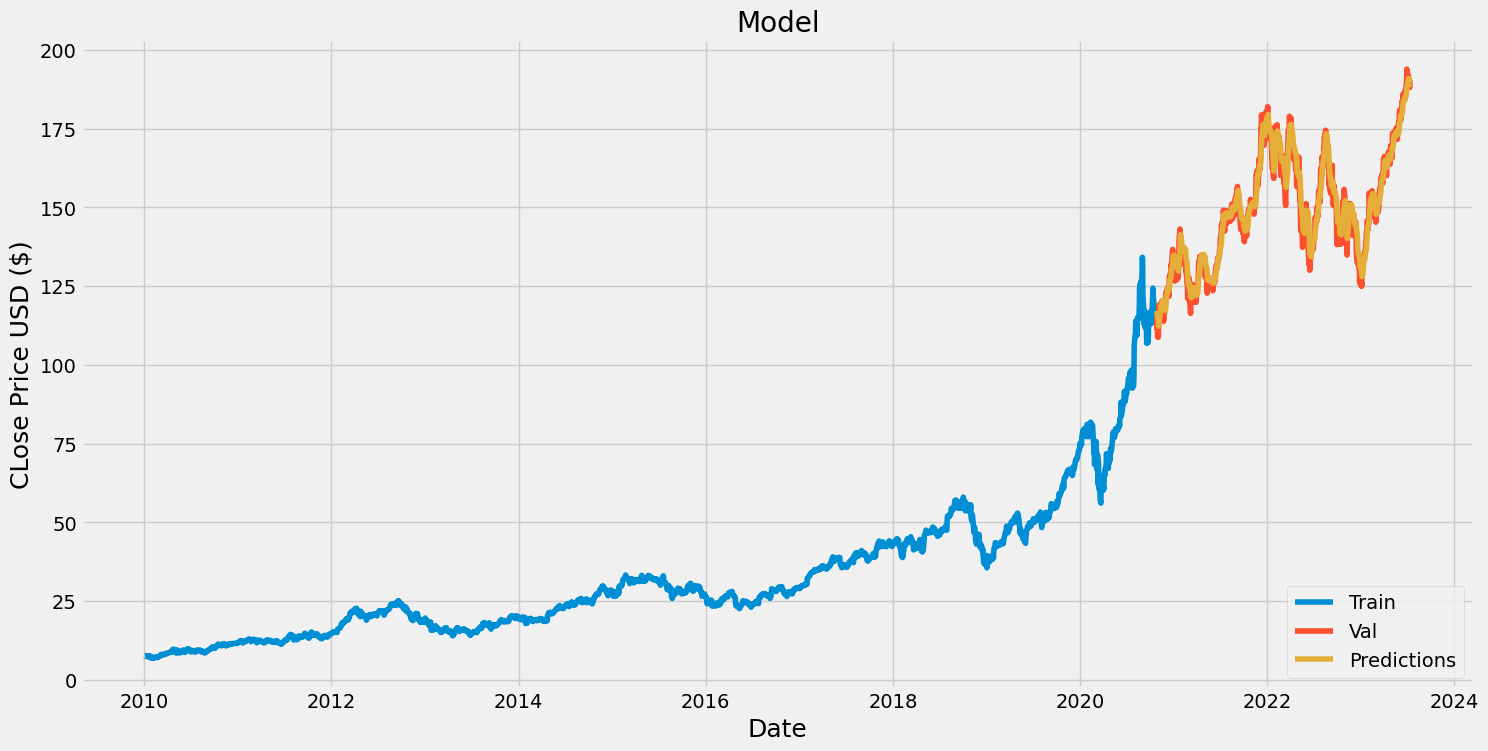


* + 1. **Experiment #4**

Number of epochs: 2

Date Range: 2010-01-01 to 2023-07-15

RMSE: 0.49



* 1. **Evaluation Metrics**

I’ve used two performance metrics, one is RMSE and the second is by visually plotting the results.

* + 1. **RMSE**

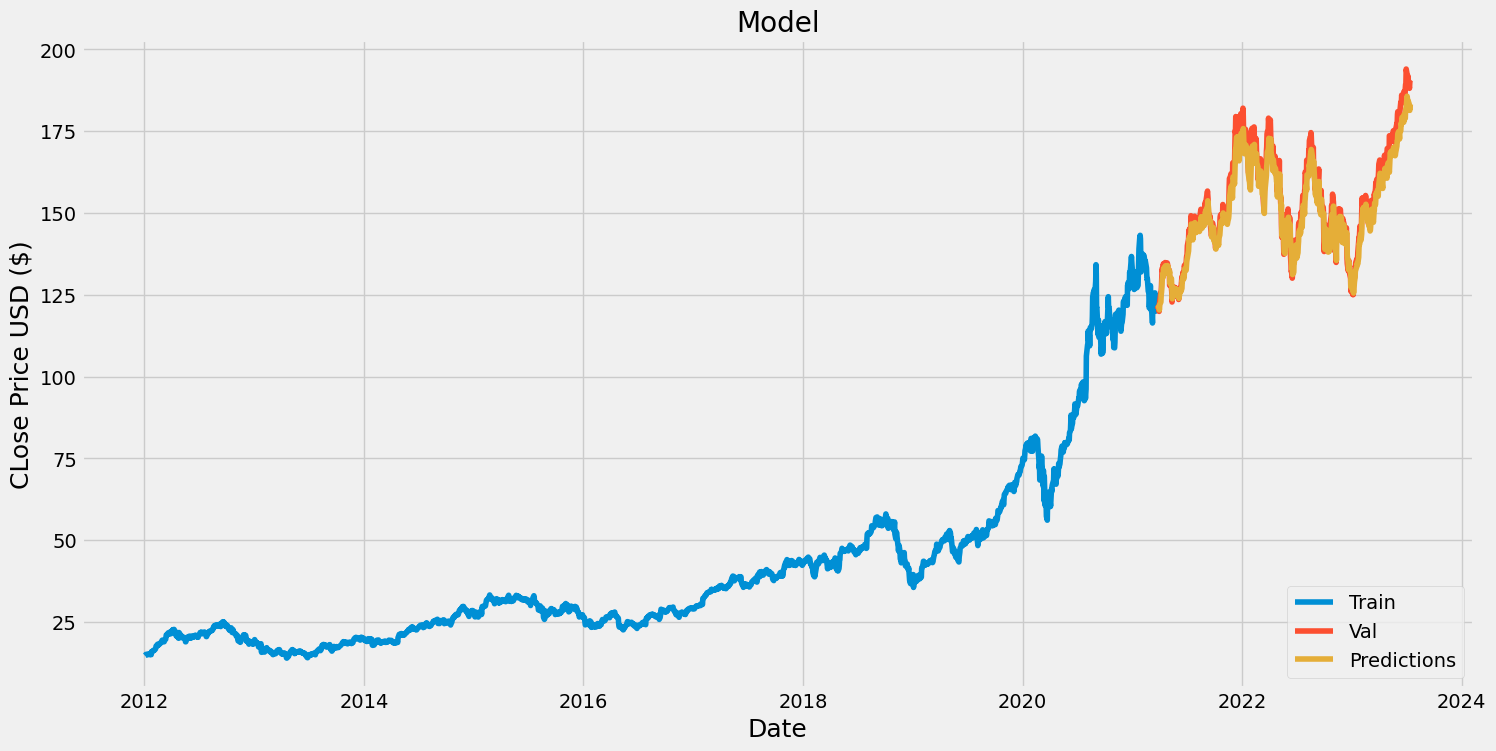
I’ve used Root Mean Squared Error to measure the accuracy of LSTM trained model, as it’s a very well recognized metric for regression tasks. RMSE calculates the square root of the average of the squared differences between the predicted values and the actual (observed) values.

Mathematically, the RMSE is calculated as follows:

RMSE = √(Σ(predicted - actual)^2 / n)

* + 1. **Plotting**

I’ve plotted the predicted value vs the real value on a time-series graph. It helps to visualize the performance. Here is one sample:



The value line represents the data which was used to train the model, the red line shows the actual price data, whereas the yellow line represents the predicated value by model. The closer the yellow and red line are, the better the model has performed.

* 1. **Experiment Settings**
* I used jupyter notebook to perform all my experiments where as, I used VS CODE to develop a Graphical User Interface as a web app.
* To calculate the RMSE score I’ve used a python library called Numpy.
* To plot the graphs and visually compare the models I’ve used python library called matplotlib.
* Whereas, to develop the python web app I’ve used streamlit.
  1. **Results**

**Table 1.** The experiment results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | Date Range | RMSE | Visual Interpretation | Conclusion |
| 10 | 2012-01-01 to 2023-07-15 | 5.26 | The lines converge but not much | The model is performing fine but could be improved. |
| 2 | 2012-01-01 to 2023-07-15 | 2.78 | The lines are converging better than the first | This is better than the last, maybe the ideal number of epoch is 2 |
| 2 | 1980-01-01 to 2023-07-15 | 15.41 | The lines are very far from each other | The worst performing model. The old data is not relevant for future prediction |
| 2 | 2010-01-01 to 2023-07-15 | 0.49 | Both the lines are almost concurrent | Provides the best result so far. |

1. Discussion

The model also performed well with an RMSE score of 0.49. It’s good, as a score of zero means the predicted and the real values were same. Changing and updating the hyperparameters helped me to achieve better model performance. There is a general notion in data science that the more the amount of data the better would be the performance of the trained model. But in my analysis it turned out to be necessarily not the case. As when I considered the data from 1980 to train the model it performed worse compared to the model which was given much less data from 2010.

The trained LSTM model is able to predict the closing price with good accuracy. I’ve also developed a Graphical User Interface with a web app. To achieve this I could have used Django or Flask but since I didn’t need any backend support I simply used streamlit. It’s a light-weight python library which made the development of web app fairly easy.

As for limitations, I’ve only considered apple stock data to evaluate my model. It may be possible that for some other stock it could perform worse. Also the model is only able to predict price for the next day. If asked for a prediction far in the future, then it will fail as it needs 60 days of data as input sequence to predict 61st day price. Also, to compare model performance I’m only using RMSE, which could be another limitation.

1. Conclusion

In conclusion, this project aimed to develop an accurate stock market prediction model utilizing Long Short-Term Memory Recurrent Neural Networks (LSTM RNNs) with data sourced from the New York Stock Exchange (NYSE).

LSTM model performed well on the sequential stock data. The trained model was able to provide output with closing price for the next day. Although these predictions weren’t always close to the actual values. But the RMSE score of 0.49 gives us a good metric that the model has been predicting values very close to the actual ones.

However, it's important to acknowledge certain limitations. The model's performance might be influenced by unpredictable market events and extreme volatility, potentially affecting its reliability during highly turbulent periods. Additionally, the model can only predict the next day’s stock price but not any further in the future.

For future scope, I can include a greater number of stocks like Microsoft, Adobe, Tesla for comparisons. That will give us a better evaluation of the model. Also, I should include more evaluation metric other than RMSE and plotting. I have already developed a GUI as a web app, in future I can try to deploy it on AWS or GCP so that it could tried and tested by communities of both computer science and finance.

References

1. "Stock Price Prediction Using LSTM, RNN and CNN-Sliding Window Model" by Sahil Mittal, N. K. Sharma, et al. (2019)

2. "A Deep Learning Framework for Financial Time Series using Stacked Autoencoders and Long Short-Term Memory" by Feng, S., Si, S., & Zhou, C. (2015)

3. "Stock Price Prediction Using Attention-based Multi-Input LSTM" by Qiu, X., Song, L., & Ni, L. M. (2019)

4. "Time Series Prediction Using LSTM Networks: Application to Stock Market" by Hundal, S. S., & Singh, D. (2018)

5. "Deep Learning for Stock Market Prediction Using Technical Indicators and Financial News Articles" by Ding, X., Zhang, Y., & Liu, T. (2015)

6. "Predicting Stock Prices Using a Feature Fusion LSTM-CNN Model" by Xie, L., Qian, W., & Zhou, Y. (2018)

7. "A Survey of Deep Learning Techniques for Stock Price Prediction" by Zhang, Y., & Wu, Y. (2019)