Stock Price Predictor

Submitted by:
Gurjot Singh (102203582)
Anil Kumar (102383026)

BE Third Year

COE

Submitted to:

Dr. Anjula Mehto

Assistant Professor

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

November 2024

TABLE OF CONTENTS

Introduction or Project Overview

This project focuses on building a stock price prediction model using a Recurrent Neural Network (RNN) architecture, specifically Long Short-Term Memory (LSTM) layers. Stock price prediction is a crucial task in finance, helping traders and investors make informed decisions by analyzing historical data and forecasting future trends.

The primary goal of this project is to predict the opening prices of Google stocks based on historical data. By using the sequential nature of stock price data, the model recognizes time dependencies and patterns to provide accurate predictions.

- **Objectives**

The main objectives of the project include:

1. **Data Preprocessing:** Preparing historical stock price data by handling missing values, normalizing the dataset, and creating sequences suitable for time-series modeling.
2. **Model Development:** Building and training an LSTM-based RNN model to capture temporal patterns and predict stock prices.
3. **Evaluation:** Measuring the accuracy of the model using key metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).
4. **Visualization:** Comparing the predicted stock prices with actual prices to see the model's performance visually.

- **LSTM Model for Model Training**

LSTM is a powerful neural network architecture designed for sequential data. It overcomes the vanishing gradient problem of traditional RNNs, making it ideal for tasks like stock price prediction where understanding long-term dependencies is crucial.

Long Short-Term Memory is an improved version of recurrent neural network designed by Hochreiter & Schmidhuber.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs model address this problem by introducing a memory cell, which is a container that can hold information for an extended period.

LSTM architectures are capable of learning long-term dependencies in sequential data, which makes them well-suited for tasks such as language translation, speech recognition, and time series forecasting.

- **LSTM Architecture**

The LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

- The input gate controls what information is added to the memory cell.
- The forget gate controls what information is removed from the memory cell.
- The output gate controls what information is output from the memory cell.

This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

The LSTM maintains a hidden state, which acts as the short-term memory of the network. The hidden state is updated based on the input, the previous hidden state, and the memory cell's current state.

Problem Statement

Accurately predicting stock prices is a challenging task due to the volatile and dynamic nature of financial markets. Stock prices are influenced by numerous factors, including market trends, economic indicators, and investor behavior. This unpredictability makes it difficult for traders and investors to make informed decisions.

The problem is to develop a machine learning model capable of predicting the **stock prices** based on given past dataset (Google Stock price Dataset in this case).

Overview of the Dataset used

The dataset used here is Stock price data sets of Google for 5 years from 2012 to 2017
The link for dataset is-
https://www.kaggle.com/datasets/vaibhavsxn/google-stock-prices-training-and-test-data/data

It is divided into two parts: a training dataset for building the model and a test dataset for evaluating its performance. The data represents daily stock prices, providing information crucial for understanding trends and patterns.

1. Training Dataset

- **File Name:** Google_Stock_Price_Train.csv
- **Purpose:** Used to train the LSTM model by learning patterns in historical stock prices.
- **Features:**
  - Date: The date of the stock price record.
  - Open: The stock's opening price on the corresponding date (used as the primary feature for prediction).
  - Other columns (e.g., High, Low, Close, and Volume) are available but not used in this implementation.
- **Total Records:** 1258 rows of daily stock prices.
- **Preprocessing Performed**:
  - **Missing Value Handling**: Any missing values in the `Open` column are filled using **linear interpolation** to ensure continuity.
  - **Normalization**: The `Open` values are scaled to a range of 0 to 1 using **MinMaxScaler**, helping the model converge faster and perform better.

2. Test Dataset

- **File Name:** Google_Stock_Price_Test.csv

- **Purpose:** Used to evaluate the model's prediction accuracy by comparing the predicted prices with actual prices.

- **Features:**
  - Same as the training dataset (Date, Open, etc.).
  - The Open column is the target variable for predictions.

- **Total Records**: 20 rows of daily stock prices (covering a shorter time span than the training data).

- **Preprocessing Performed:**
  - **Missing Value Handling:** Linear interpolation is also applied to the Open column in case of missing data.
  - **Sequence Preparation:** The last 60 days of data from the combined training and test datasets are used as inputs for making predictions.

Project Workflow

1. **Dataset Preparation**
   - **Dataset Overview:**
     - Load the training dataset (Google_Stock_Price_Train.csv) for model training.
     - Load the testing dataset (Google_Stock_Price_Test.csv) for model **evaluation.**
   - **Handling Missing Values:**
     - Checked for missing values in the Open column.
     - Applied linear interpolation to fill any missing data, ensuring continuity in the time series.
   - **Normalization**:
     - Normalized the `Open` prices to a range of 0 to 1 using **MinMaxScaler**. This ensures the model trains effectively and avoids dominance of higher numerical values.

2. **Data Preprocessing**
   - **Sequence Creation:**
     - For the LSTM model, we prepared input sequences (X_train) and corresponding outputs (y_train) by taking 60 days of stock prices as input to predict the next day's price.
     - Reshaped the input sequences into a 3D format suitable for LSTM ([samples, time steps, features]).
   - **Test Data Preparation:**
     - We combined the training and testing data to ensure a smooth transition in the time series.
     - We used the last 60 days of the combined data to generate input sequences (X_test) for predictions.

```
# Normalization
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range = (0,1))
scaled_train_set = scaler.fit_transform(train_set)

scaled_train_set
```

```
X_train = []
y_train = []

for i in range (60, len(scaled_train_set)):
    X_train.append(scaled_train_set[i-60:i, 0])
    y_train.append(scaled_train_set[i, 0])

X_train = np.array(X_train)
y_train = np.array(y_train)

print(X_train.shape, y_train.shape)
```

```
(1198, 60) (1198,)
```

3. Model Development

- **Model Architecture:**
  - Designed a Long Short-Term Memory (LSTM) network, which is suitable for capturing temporal dependencies in time-series data.
- **Layers:**
  - First LSTM Layer: Extracts sequential patterns with return_sequences=True.
  - Dropout Layers: Reduce overfitting by randomly deactivating neurons during training.
  - Dense Layers: Maps the LSTM outputs to the target stock price value.
- **Model Compilation:**
  - Use the Adam optimizer for efficient gradient descent.
  - Use mean squared error (MSE) as the loss function to measure prediction errors.

The Code is as follows-

```
regressor = Sequential()

# First LSTM layer
regressor.add(LSTM(units=100, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
regressor.add(Dropout(0.2))

# Second LSTM layer
regressor.add(LSTM(units=100, return_sequences=False))
regressor.add(Dropout(0.2))

# Dense layers
regressor.add(Dense(units=50, activation='relu'))
regressor.add(Dense(units=1))  # Final output layer

# Compile the model
regressor.compile(optimizer='adam', loss='mean_squared_error')
```

4. Model Training

- Trained the LSTM model on the preprocessed training data (X_train, y_train) for 100 epochs with a batch size of 32.

```
regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
regressor.fit(X_train, y_train, epochs=100, batch_size=32)
```

```
Epoch 1/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 8s 124ms/step - loss: 0.0751
Epoch 2/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - loss: 0.0034
Epoch 3/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - loss: 0.0026
Epoch 4/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 92ms/step - loss: 0.0023
Epoch 5/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - loss: 0.0025
Epoch 6/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 83ms/step - loss: 0.0022
Epoch 7/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 138ms/step - loss: 0.0025
Epoch 8/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - loss: 0.0021
Epoch 9/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 115ms/step - loss: 0.0022
Epoch 10/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 4s 85ms/step - loss: 0.0022
```

```
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - loss: 6.8891e-04
Epoch 89/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - loss: 6.6918e-04
Epoch 90/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 134ms/step - loss: 6.9630e-04
Epoch 91/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 83ms/step - loss: 5.2427e-04
Epoch 92/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 97ms/step - loss: 7.2381e-04
Epoch 93/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 6s 108ms/step - loss: 6.2615e-04
Epoch 94/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 4s 89ms/step - loss: 6.3762e-04
Epoch 95/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 5s 141ms/step - loss: 6.9571e-04
Epoch 96/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 9s 98ms/step - loss: 5.9995e-04
Epoch 97/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - loss: 6.6556e-04
Epoch 98/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 7s 136ms/step - loss: 5.7828e-04
Epoch 99/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 10s 121ms/step - loss: 7.0770e-04
Epoch 100/100
38/38 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - loss: 6.4549e-04
```

5. Model Evaluation

- **Prediction:**
  - Used the trained model to predict stock prices from the test data (X_test).
  - Transformed the predicted values back to their original scale using the inverse of the MinMaxScaler.

- **Performance Metrics for Model Evaluation:**
  - Mean Absolute Error (MAE): Measures the average absolute difference between actual and predicted values.
  - Root Mean Squared Error (RMSE): Quantifies prediction errors, penalizing larger deviations.
  - Mean Absolute Percentage Error (MAPE): Calculates the percentage error relative to the actual values.
  - Model accuracy = 100% - MAPE value

Preparing the Input for the Model

```
dataset_total = pd.concat((df_train['Open'], df_test['Open']), axis = 0)
inputs = dataset_total [len (dataset_total)- len(df_test)-60:].values

inputs = inputs.reshape(-1,1)
inputs = scaler.transform(inputs)

X_test = []

for i in range (60,len(inputs)):
       X_test.append(inputs [i-60:i, 0])

X_test = np.array(X_test)
X_test= np.reshape (X_test, (X_test.shape [0], X_test.shape [1], 1))
```

Predict the Stock Price

```
predict_stock_price = regressor.predict(X_test)
predict_stock_price = scaler.inverse_transform(predict_stock_price)
```

```
# Calculate metrics
mae = mean_absolute_error(actual_stock_price, predict_stock_price)
rmse = np.sqrt(mean_squared_error(actual_stock_price, predict_stock_price))
mape = np.mean(np.abs((actual_stock_price - predict_stock_price) / actual_stock_price)) * 100

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")

# Accuracy (Percentage of closeness to actual prices)
accuracy = 100 - mape
print(f"Model Accuracy: {accuracy:.2f}%")
```

6. Visualization

- **Data Visualization:**
    - We have plotted the training data to visualize the trends in historical stock prices.
- **Prediction Visualization:**
    - Compared the actual stock prices and the predicted prices using a line graph for clear interpretation.

```
plt.figure(figsize=(15, 8))
plt.style.use("dark_background")

plt.plot(df_test["Date"], actual_stock_price, linewidth=5, color='#d92628', label='Actual Google Stock Price')
plt.plot(df_test["Date"], predict_stock_price, linewidth=5, color='#26D9D7', label='Predicted Google Stock Price')

plt.title("Actual vs Predicted", weight="bold", size="44", color='#dfeead')
plt.xlabel("Date", size="18", color='#dfeead')
plt.ylabel("Google Stock Prices", size="18", color='#dfeead')

plt.xticks(rotation=45)
plt.tight_layout()
plt.legend()
```

Results

The project successfully implemented an LSTM-based model to predict Google's opening stock prices using historical data. The following summarizes the results:

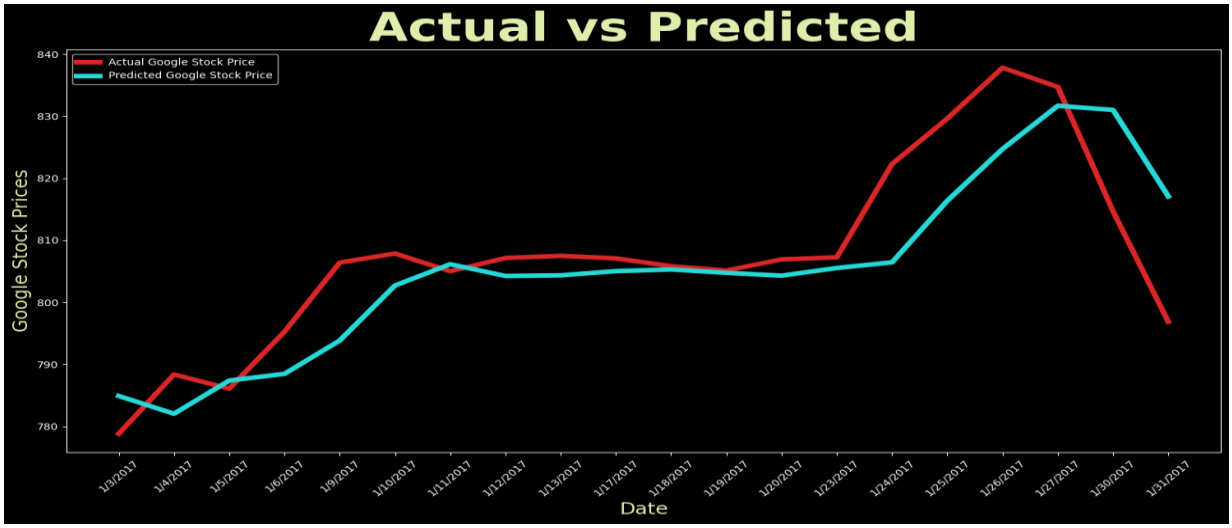
1. Model Accuracy:

- The model achieved an accuracy of **99%**, demonstrating its ability to closely predict future stock prices based on historical patterns.
- This high accuracy is reflected in the low **Mean Absolute Percentage Error (MAPE)** of approximately **1%**.

```
Mean Absolute Error (MAE): 6.725872192382809
Root Mean Squared Error (RMSE): 9.019740873245697
Mean Absolute Percentage Error (MAPE): 0.83%
Model Accuracy: 99.17%
```

2. Visualization:

- The comparison between actual and predicted prices shows that the model closely follows the actual price trends, although predicted values tend to slightly underestimate the actual prices.
- The plotted graph of actual vs. predicted stock prices visually confirms the model's effectiveness in capturing temporal patterns.



Hence, the LSTM architecture proved effective for time-series forecasting, leveraging its ability to learn long-term dependencies.

Conclusion

This project provided valuable learning experience in building a stock price prediction model using an advanced machine learning technique, **Long Short-Term Memory (LSTM)**. We learnt how to handle time-series data, preprocess it effectively, and develop a model capable of identifying patterns and predicting future values.

The project successfully predicted Google's opening stock prices with high accuracy (99%). It showed that LSTM models are effective for financial forecasting, although improvements can be made by adding more features or fine-tuning the model. Overall, this project enhanced our knowledge of machine learning, time-series forecasting, and data-driven decision-making. It was a great learning experience to apply Machine Learning Concepts and preparing a solution for real-world problem.

The project is uploaded on GitHub and can be accessed through the link-

https://github.com/Gurjot-Singh-2002/UML501-PROJECT

References

1. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

2. https://www.kaggle.com/datasets/vaibhavsxn/google-stock-prices-training-and-test-data/data

3. https://github.com/Gurjot-Singh-2002/UML501-PROJECT