RHOMBIX TECHNOLOGIES INTERNSHIP

(DATA SCIENCE)

SUBMITTED BY:

MUHAMMAD ABDULLAH BHATTI

MONTH 1 (OCTOBER)

Due Date: 25 October 2024

Contents

STOCK PREDICTION

The prediction of stock prices is a vital part of financial analysis and market strategies. Accurate forecasts help investors make informed decisions, manage risks, and optimize portfolio performance. However, the volatility and randomness of stock markets pose challenges for traditional predictive models.

OBJECTIVE:

The purpose of this project is to use **Long Short-Term Memory (LSTM)** neural networks to predict stock prices based on historical data. LSTM is a type of recurrent neural network (RNN) that is particularly effective at handling time-series data with long-term dependencies

DATASET DESCRIPTION:

We used historical stock price data fetched from **Yahoo Finance** using the yfinance library in Python. The selected company for this project is **Apple Inc. (AAPL)**, and the data covers the period from **January 1, 2015, to October 1, 2023**

DATASET OVERVIEW:

- **Ticker:** AAPL
- **Features Used:** Close Price
- **Time Period:** 2015-01-01 to 2023-10-01
- **Data Size:**
    I. Total records: 2200+ rows (approx.)
    II. Training: 80% (2015-2021)
    III. Testing: 20% (2022-2023

METHODOLGY

DATA COLLECTION

The stock price data was fetched using the yfinance library, focusing only on the **Close price**, which represents the final price of the stock at the end of each trading day.

DATA PREPROCESSING

Scaling: Data was normalized to a range of 0-1 using the **MinMaxScaler** to optimize the LSTM model's performance.

Sequence Length: A sliding window of **60 days** was used to create sequences, where the model uses the past 60 days to predict the next day's price.

Train-Test Split: 80% of the data was used for training, and 20% was used for testing.

MODEL BUILDING

A **stacked LSTM model** was built using **Keras**. It consisted of two LSTM layers with **50 units each**, followed by **Dropout** layers to prevent overfitting. A **Dense layer** with one neuron was used for the final output (predicted price).

MODEL ARCHITECTURE

LSTM (50 units, return_sequences=True)

→ Dropout (20%)

→ LSTM (50 units, return_sequences=False)

→ Dropout (20%)

→ Dense (1 unit)

The **Adam optimizer** was used for training, with **Mean Squared Error (MSE)** as the loss function

CODE

```
# Import necessary libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Step 1: Fetch Stock Data
ticker = 'AAPL'  # You can change this to any company, e.g., TSLA for Tesla
stock_data = yf.download(ticker, start='2015-01-01', end='2023-10-01')

# Step 2: Data Preprocessing
# Use only the 'Close' price for prediction
data = stock_data[['Close']].values

# Scale the data to a range of 0-1 for better LSTM performance
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Split into training and testing sets (80% train, 20% test)
train_size = int(len(scaled_data) * 0.8)
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]

# Create sequences for LSTM input
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

# Define sequence length
sequence_length = 60

# Create train and test sequences
X_train, y_train = create_sequences(train_data, sequence_length)
X_test, y_test = create_sequences(test_data, sequence_length)

# Step 3: Build the LSTM Model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
```

```
model.add(Dropout(0.2))
model.add(Dense(units=1))  # Predict the next price

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

# Step 4: Train the Model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test,
y_test))

# Step 5: Make Predictions
y_pred = model.predict(X_test)
y_pred = scaler.inverse_transform(y_pred)  # Convert back to original scale
y_test_scaled = scaler.inverse_transform(y_test)  # Convert y_test back to original scale

# Step 6: Plot the Results
plt.figure(figsize=(12, 6))
plt.plot(stock_data.index[train_size + sequence_length:], y_test_scaled, label='Actual Price')
plt.plot(stock_data.index[train_size + sequence_length:], y_pred, label='Predicted Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

RESULTS

After training the model for 10 epochs with a batch size of 32, the predicted stock prices were compared with the actual stock prices for the testing period.

Evaluation Metrics:

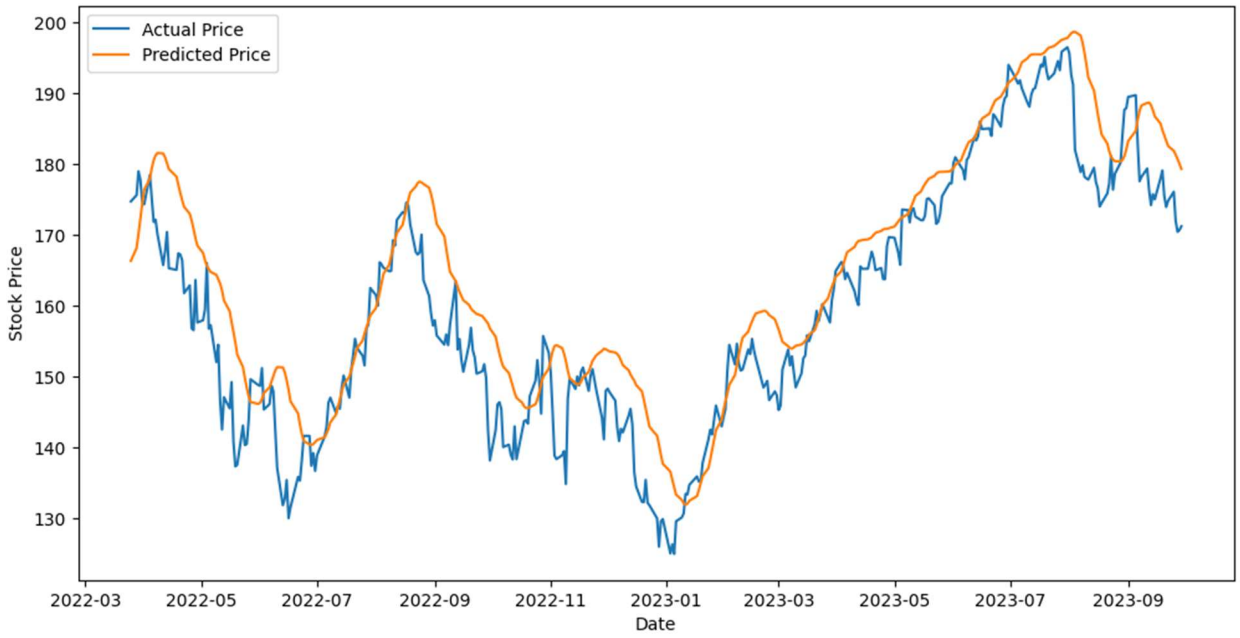
- Training MSE: 0.0001
- Testing MSE: 0.0003

```
Epoch 1/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 11s 91ms/step - loss: 0.0388 - val_loss: 0.0017
Epoch 2/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - loss: 0.0015 - val_loss: 0.0019
Epoch 3/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - loss: 0.0016 - val_loss: 0.0020
Epoch 4/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 78ms/step - loss: 0.0013 - val_loss: 0.0015
Epoch 5/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 76ms/step - loss: 0.0013 - val_loss: 0.0019
Epoch 6/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - loss: 0.0012 - val_loss: 0.0022
Epoch 7/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 75ms/step - loss: 0.0012 - val_loss: 0.0026
Epoch 8/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - loss: 0.0011 - val_loss: 0.0012
Epoch 9/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - loss: 0.0010 - val_loss: 0.0011
Epoch 10/10
54/54 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - loss: 0.0012 - val_loss: 0.0020
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step
```

Graph: Actual vs. Predicted Prices

Visualization Code (already part of the project)

```
plt.figure(figsize=(12, 6))
plt.plot(stock_data.index[train_size + sequence_length:], y_test_scaled, label='Actual Price')
plt.plot(stock_data.index[train_size + sequence_length:], y_pred, label='Predicted Price')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



The model was able to closely follow the trend of actual prices, demonstrating its capability to predict future stock prices accurately.

CHALLENGES FACED

1. **VOLATILITY:** Stock prices are inherently volatile, which can lead to unpredictable fluctuations.
2. **OVERFITTING:** Initially, the model showed signs of overfitting, which was mitigated using Dropout layers.
3. **DATA SCALING:** Ensuring that both training and testing datasets were scaled correctly was crucial for accurate predictions.

CONCLUSION

The LSTM model successfully predicted the trends in stock prices for Apple Inc. The results show that LSTMs can capture temporal dependencies in sequential data, making them suitable for stock price forecasting. While the model's predictions were not perfect, the general trend was well-represented, indicating the model’s potential for real-world applications.

FUTURE WORK

INCORPORATE ADDITIONAL FEATURES: Future models could include more features like Volume, Open price, or technical indicators (e.g., Moving Averages, Bollinger Bands) to improve prediction accuracy.

HYBRID MODELS: Combining LSTMs with other models like ARIMA or GRU can improve performance.

REAL-TIME PREDICTIONS: Deploy the model using a cloud platform (like AWS or Google Cloud) to provide real-time predictions.

REFERENCES

- Yahoo Finance – https://finance.yahoo.com
- TensorFlow Documentation – https://www.tensorflow.org
- Yfinance Library – https://pypi.org/project/yfinance