lstm-full-code

January 5, 2025

[1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from tensorflow.keras.models import load_model
```

[2]:
```
def get_stock_data(ticker, start_date, end_date):
    """
    Fetch stock data from Yahoo Finance
    """
    try:
        stock = yf.Ticker(ticker)
        df = stock.history(start=start_date, end=end_date)
        return df
    except Exception as e:
        print(f"Error fetching data for {ticker}: {e}")
        return None
```

[3]:
```
def prepare_data(df, look_back=60, split_ratio=0.8):
    """
    Prepare data for LSTM model
    """
    # Select 'Close' prices and convert to numpy array
    data = df['Close'].values.reshape(-1, 1)

    # Scale the data
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data)

    # Create sequences for LSTM
    X, y = [], []
    for i in range(look_back, len(scaled_data)):
        X.append(scaled_data[i-look_back:i, 0])
        y.append(scaled_data[i, 0])
```

```
    X, y = np.array(X), np.array(y)

    # Reshape X to match LSTM input shape [samples, time steps, features]
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    # Split into train and test sets
    train_size = int(len(X) * split_ratio)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]

    return X_train, X_test, y_train, y_test, scaler
```

[4]:
```
def create_lstm_model(look_back):
    """
    Create and compile LSTM model
    """
    model = Sequential([
        LSTM(units=50, return_sequences=True, input_shape=(look_back, 1)),
        Dropout(0.2),
        LSTM(units=50, return_sequences=True),
        Dropout(0.2),
        LSTM(units=50),
        Dropout(0.2),
        Dense(units=1)
    ])

    model.compile(optimizer='adam', loss='mean_squared_error')
    model.summary()
    return model
```

[5]:
```
# 4. Train and Evaluate Model
def train_and_evaluate(model, X_train, X_test, y_train, y_test, scaler,␣
 ↪epochs=50, batch_size=32):
    """
    Train the model and make predictions
    """
    # Train the model
    history = model.fit(
        X_train, y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_split=0.1,
        verbose=1
    )

    # Save the model
```

```
    print(f"Saving the model to lstm_model.h5...")
    model.save("lstm_model.h5")
    return history

# 5. Load Model and Use for Predictions
def load_and_predict(model_path, X_train, X_test, y_train, y_test, scaler):
    """
    Load the saved model and make predictions on train and test data.
    """
    # Load the saved model
    print(f"Loading the model from {model_path}...")
    model = load_model(model_path)

    # Make predictions
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    # Inverse transform predictions
    train_predict = scaler.inverse_transform(train_predict)
    y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
    test_predict = scaler.inverse_transform(test_predict)
    y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))

    return train_predict, test_predict, y_train_inv, y_test_inv
```

[9]:
```
def plot_results(df, train_predict, test_predict, look_back, split_ratio):
    """
    Plot actual vs predicted stock prices
    """
    # Create figure and axis objects
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 12))

    # Determine the split point
    train_size = int(len(df) * split_ratio)

    # Adjust indices for training predictions
    train_predict_index = df.index[look_back:train_size]  # Ensure the length␣
 ↪matches train_predict
    ax1.plot(train_predict_index, df['Close'].values[look_back:train_size],␣
 ↪label='Actual Price', color='blue')
    ax1.plot(train_predict_index, train_predict.flatten()[:
 ↪len(train_predict_index)], label='Predicted Price', color='red')  # Slice␣
 ↪the prediction
    ax1.set_title('Stock Price Prediction - Training Data')
    ax1.set_xlabel('Date')
    ax1.set_ylabel('Price')
    ax1.legend()
```

```
    # Adjust indices for testing predictions
    test_predict_index = df.index[train_size:]  # Ensure the length matches␣
↪test_predict
    ax2.plot(test_predict_index, df['Close'].values[train_size:], label='Actual␣
↪Price', color='blue')
    ax2.plot(test_predict_index[:len(test_predict)], test_predict.flatten(),␣
↪label='Predicted Price', color='red')  # Slice the prediction
    ax2.set_title('Stock Price Prediction - Testing Data')
    ax2.set_xlabel('Date')
    ax2.set_ylabel('Price')
    ax2.legend()

    plt.tight_layout()
    plt.show()
```

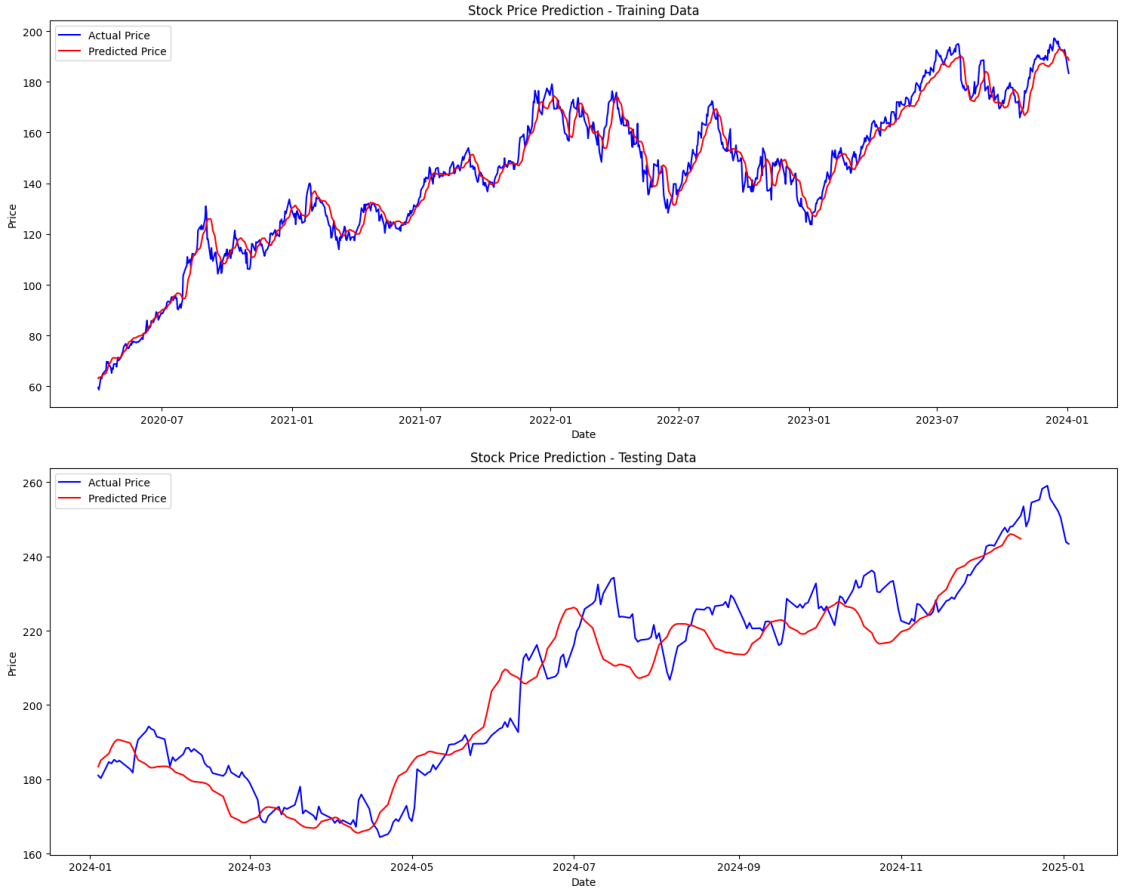
[7]:
```
def plot_loss(history):
    """
    Plot training and validation loss
    """
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

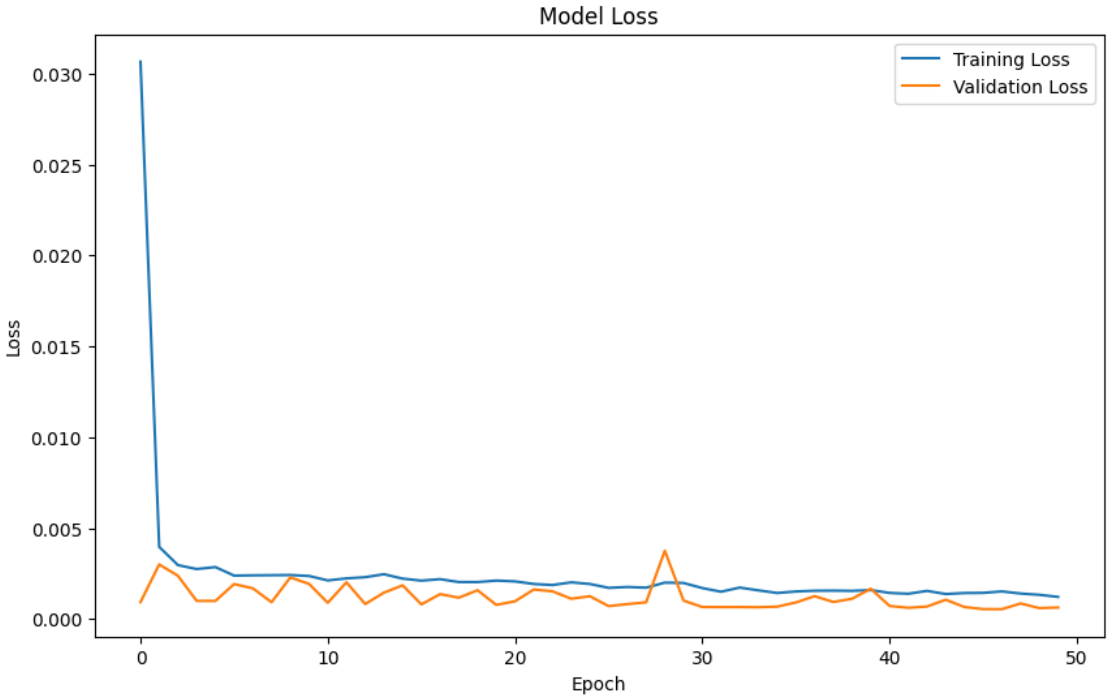
[10]:
```
import os
import tensorflow as tf
from datetime import datetime, timedelta

if __name__ == "__main__":
    # Set parameters
    TICKER = "AAPL"  # Stock ticker
    LOOK_BACK = 60    # Number of previous days to use for prediction
    SPLIT_RATIO = 0.8
    EPOCHS = 50
    BATCH_SIZE = 32
    MODEL_PATH = "lstm_model.h5"  # Path to save/load the model

    # Get dates for the last 5 years
    end_date = datetime.now()
    start_date = end_date - timedelta(days=5 * 365)

    # 1. Get stock data
```

```
    print(f"Fetching {TICKER} stock data...")
    df = get_stock_data(TICKER, start_date, end_date)

    # 2. Prepare data
    print("Preparing data...")
    X_train, X_test, y_train, y_test, scaler = prepare_data(df, LOOK_BACK,␣
↪SPLIT_RATIO)

    # 3. Train or load model
    if os.path.exists(MODEL_PATH):
        print(f"Loading model from {MODEL_PATH}...")
        model = tf.keras.models.load_model(MODEL_PATH)
    else:
        print("Creating and training a new LSTM model...")
        model = create_lstm_model(LOOK_BACK)

        # Train and save the model
        history = train_and_evaluate(model, X_train, X_test, y_train, y_test,␣
↪scaler, EPOCHS, BATCH_SIZE)

    # 4. Make predictions
    print("Making predictions...")
    train_predict, test_predict, y_train_inv, y_test_inv = load_and_predict(
        MODEL_PATH, X_train, X_test, y_train, y_test, scaler
    )

    # 5. Visualize results
    print("Plotting results...")
    plot_results(df, train_predict, test_predict, LOOK_BACK, SPLIT_RATIO)
    plot_loss(history)  # Pass the history object here

    # 6. Calculate and display performance metrics
    from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪r2_score

    mse = mean_squared_error(y_test_inv.flatten(), test_predict.flatten())
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test_inv.flatten(), test_predict.flatten())
    r2 = r2_score(y_test_inv.flatten(), test_predict.flatten())

    print("\nModel Performance Metrics (Test Set):")
    print(f"Root Mean Squared Error: ${rmse:.2f}")
    print(f"Mean Absolute Error: ${mae:.2f}")
    print(f"R-squared Score: {r2:.4f}")
```

```
Fetching AAPL stock data…

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
```

```
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

Preparing data…
Loading model from lstm_model.h5…
Making predictions…
Loading the model from lstm_model.h5…

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

30/30              1s 32ms/step
8/8              0s 16ms/step
Plotting results…
```






```
Model Performance Metrics (Test Set):
Root Mean Squared Error: $7.95
Mean Absolute Error: $6.43
R-squared Score: 0.9058
```

[ ]: